NPS52-88-014

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

A GRAPHICS FACILITY
FOR INTEGRATION, EDITING, AND DISPLAY
OF SLOPE, CURVATURE, AND CONTOURS
FROM A DIGITAL TERRAIN ELEVATION DATABASE

by

Dennis G. Felhoelter

June 1988

Thesis Advisor: Robert B. McGhee

Prepared for:
USACDEC
Ft. Ord, California 93941

NAVAL POSTGRADUATE SCHOOL
Monterey, California

Rear Admiral R. C. Austin
Superintendent

Kneale T. Marshall
Acting Provost

This thesis is prepared in conjunction with research sponsored in part by contract from the United States Army Combat Developments Experimentation Center (USACDEC) under MIPR ATEC 88-86.

The issuance of this thesis as a technical report is concurred by:

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; Distribution is unlimited.
2b DECLASSIFICATION / DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) NPS52-88-014	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Naval Postgraduate School 6b. OFFICE SYMBOL (*If applicable*) Code 52	7a. NAME OF MONITORING ORGANIZATION Naval Postgraduate School
6c. ADDRESS (*City, State, and ZIP Code*) Monterey, California 93943-5000	7b. ADDRESS (*City, State, and ZIP Code*) Monterey, California 93943-5000
8a. NAME OF FUNDING / SPONSORING ORGANIZATION USACDEC 8b. OFFICE SYMBOL (*If applicable*)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER ATEC 88-86
8c. ADDRESS (*City, State, and ZIP Code*) Ft. Ord, California 93941	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO / WORK UNIT ACCESSION NO.

11. TITLE (*Include Security Classification*)
A GRAPHICS FACILITY FOR INTEGRATION, EDITING, AND DISPLAY OF SLOPE, CURVATURE, AND CONTOURS FROM A DIGITAL TERRAIN ELEVATION DATABASE

12. PERSONAL AUTHOR(S)
Felhoelter, Dennis G.

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (*Year, Month, Day*)	15 PAGE COUNT
Master's Thesis	FROM _____ TO _____	1988 June	118

16. SUPPLEMENTARY NOTATION
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

17 COSATI CODES			18. SUBJECT TERMS (*Continue on reverse if necessary and identify by block number*)
FIELD	GROUP	SUB-GROUP	Terrain Classification; Route Planning; B-spline Smoothing

19. ABSTRACT (*Continue on reverse if necessary and identify by block number*)

The ability of man to scan terrain, compare it with a topographical representation, and make decisions based on his analysis is a unique and complex talent. Teaching a machine to make these same comparisons and analyses is a formidable task. However, the development of acceptable algorithms to permit the appropriate classifications of terrain will expand the capabilities of machines in a number of endeavors including route planning and movement across selected terrain.

Recent research in terrain classification has centered around using mathematical equations to represent small cells of land. This thesis will attempt to improve the classification of terrain data by expanding the type of information available, and by enhancing the quality of the data through the use of a graphics tool (bicubic splines) to edit and smooth this raw elevation data for more accurate elevation representation.

20 DISTRIBUTION / AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION Unclassified
22a NAME OF RESPONSIBLE INDIVIDUAL Professor Robert B. McGhee	22b. TELEPHONE (*Include Area Code*) (408) 646-2095 22c. OFFICE SYMBOL Code 52Mz

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted.
All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE

☆ U.S. Government Printing Office: 1986—606-24

Approved for public release; distribution is unlimited.

A Graphics Facility for Integration, Editing, and Display of Slope, Curvature, and Contours from a Digital Terrain Elevation Database

by

Dennis G. Fellhoelter
Major, United States Marine Corps
B.S., University of Louisville, 1972

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL

June 1988

ABSTRACT

The ability of man to scan terrain, compare it with a topographical representation, and make decisions based on his analysis is a unique and complex talent. Teaching a machine to make these same comparisons and analyses is a formidable task. However, the development of acceptable algorithms to permit the appropriate classifications of terrain will expand the capabilities of machines in a number of endeavors including route planning and movement across selected terrain.

Recent research in terrain classification has centered around using mathematical equations to represent small cells of land. This thesis will attempt to improve the classification of terrain data by expanding the type of information available, and by enhancing the quality of the data through the use of a graphics tool (bicubic splines) to edit and smooth this raw elevation data for more accurate elevation representation.

TABLE OF CONTENTS

I. INTRODUCTION 1
A. BACKGROUND 1
B. ORGANIZATION OF CHAPTERS 3
C. PURPOSE 4
II. SURVEY OF PREVIOUS WORK 5
A. INTRODUCTION 5
B. SURFACE CLASSIFICATION 6
1. Basic Concepts 6
a. Surface Storage Methods 6
b. Terrain Types 8
c. Locating Hills and Basins 8
(1) Sorting 9
(2) Hill Climbing 9
(3) Use of Slope Lines 9
(4) Local Procedures 10
d. Thinning of Results 10
e. Junction Points 11
f. Model Creation 11
g. Results of Work 11
2. Cubic Approximation to Terrain 12
a. Basic Model 12
b. Mathematical Basis 12
c. Mathematical Values 13
3. Classification Based on Slope and Curvature 14
C. TERRAIN CLASSIFICATION 16
D. TERRAIN SMOOTHING 18
1. Spatial Filtering 18
2. Planar Patches 18
3. Graphics Representations 19
a. Bezier Patches 19
b. Cardinal Patches 20
c. B-spline Patches 20
E. ROUTE PLANNING 21
F. SUMMARY 21

III. DETAILED PROBLEM STATEMENT 23
A. INTRODUCTION 23
B. HUNTER LIGGETT DATA 24
1. Resolution of Data 24
2. Accuracy of Data in Database 25
a. Hilltop Smoothing 25
b. Contour Crossing in Flat Terrain 25
C. B-SPLINE SMOOTHING 26
D. HARDWARE ENVIRONMENT 29
E. SUMMARY 29
IV. ALGORITHMS DEVELOPED 31
A. INTRODUCTION 31
B. TERRAIN CLASSIFICATION ALGORITHMS 31
1. Normalization of Data 31
2. Negative Discriminant 32
3. Default Classifications 33
4. R-line Tests Not Shown 33
5. Rewrite of Conditional Loops 34
C. CONTOUR ALGORITHMS 34
1. Contour Intervals 34
2. Vegetation 35
D. B-SPLINE SMOOTHING ALGORITHMS 35
E. SLOPE ALGORITHMS 36
F. DRAWING ALGORITHMS 37
G. USER'S GUIDE 37
1. Creating Terrain Elevation/Vegetation Data 37
2. Loading the System 40
3. Getting Started 40
4. Running the System 41
a. Smoothing the Terrain 42
b. Terrain Classification 42
c. Displaying Contours 43
d. Slope Classification 43
e. Drawing Functions 44
H. SUMMARY 45
V. EXPERIMENTAL RESULTS 46
A. INTRODUCTION 46
B. TYPICAL DISPLAYS 46
1. Classification Display 47

2. Slope Display 48
3. Contour Display 51
4. Vegetation Display 54
C. COMPARISON OF DISPLAYS 56
1. Smoothing of Elevation Data 56
2. Classification 56
3. Threshold Adjustments 60
D. SUMMARY 65
VI. SUMMARY AND CONCLUSIONS 66
A. RESEARCH CONTRIBUTIONS 66
B. RESEARCH EXTENSIONS 66
LIST OF REFERENCES 69
APPENDIX A - CONVERSION FUNCTIONS 70
APPENDIX B - TERRAIN CONSTANTS 71
APPENDIX C - TERRAIN CLASSIFICATION FUNCTIONS 73
APPENDIX D - SLOPE CLASSIFICATION FUNCTIONS 82
APPENDIX E - CONTOUR CLASSIFICATION FUNCTIONS 84
APPENDIX F - B-SPLINE SMOOTHING FUNCTIONS 88
APPENDIX G - GRAPHICS DRAWING FUNCTIONS 93
INITIAL DISTRIBUTION LIST 107

LIST OF FIGURES

Figure 2.1	Sample Terrain	14
Figure 3.1	B-Spline Control Points	28
Figure 5.1	Typical Terrain Classification and Slope Display - fq5580	50
Figure 5.2	Typical Contour and Vegetation Displays - fq5580	53
Figure 5.3	Unsmoothed and Smoothed Contour Displays - fq5886	57
Figure 5.4	Unsmoothed and Smoothed Classification Displays - fq5886	58
Figure 5.5	Smoothed Classification Display - Curve 0.004 - fq5886	61
Figure 5.6	Smoothed Classification Display - Curve 0.006 - fq5886	62
Figure 5.7	Five kilometer square Classification Display - fq5685	64

I. INTRODUCTION

A. BACKGROUND

Human beings are uniquely adept at scanning the surrounding environment and making decisions based on their analysis of the data collected visually. In a typical military environment, one type of information that is often collected in this manner is terrain data. When compared to information on a standard topographical map, both tactical and administrative decisions can be made based on the individual's analysis of the expected and observed data. Teaching a machine to scan and make these same decisions is a complex and challenging task. While man is more adept at making decisions based on a variety of parameters (i.e., tactical, physical, administrative, etc.), he is not as capable of analyzing and storing the thousands of pieces of information available to him concerning large areas of terrain and typically relies on some form of medium for storage of this information (i.e., maps). A machine, on the other hand, can store and analyze millions of pieces of data with a high degree of accuracy, but is slow and unreliable at making decisions when confronted with a number of parameters and rules including complex, objective military requirements. With this in mind, this thesis attempts to improve and advance the state of the art of storage and manipulation of dense elevation data, to enhance terrain classification and recognition. The ultimate goal is to integrate this capability in a system to provide advanced route planning capabilities for either manned or autonomous vehicles.

The storage and manipulation of vast amounts of terrain data is an area that is already being intensively investigated. The Defense Mapping Agency (DMA), which is responsible for maintaining accurate cartographic maps for the entire world, is currently working on a system that will computerize the thousands of maps that are primarily

maintained by hand. Their system, which is planned for a production environment in the early 1990's, will mainly be capable of scanning and storing data about a particular map one pixel at a time, although work is also proceeding on the use of digital terrain databases. DMA is not alone in this work. The National Geographic Society, which also maintains a large number of maps for its members, is attempting to computerize the maintenance and update of its maps. In both cases, the volume of data is immense and the manual effort is just as large. [Ref. 1]

The use of digital elevation data can be a major step forward in improving the capability to maintain maps and other assorted information about the world land masses. When this is done, the problem of storage, manipulation, and display of realistic, dense, and accurate elevation data becomes a problem of paramount importance. If the data which is stored can be utilized to recreate specific information about certain areas, a major stride will have been made in further computerizing the work of mapping and display of data about selected areas of terrain.

Programming a machine to extract digitized data from a database and analyze and classify individual cells of terrain data is one of the first steps in developing a fully autonomous vehicle. With this capability, the difficult process of teaching a machine to think like a human in a given tactical situation can begin. However, the sheer magnitude of the problem, storing data in a computer to represent a typical grid square, can be overwhelming. New and more powerful computers provide the capability to store and manipulate increased volumes of data. This enables researchers to explore new areas in a drive to eliminate paper maps and charts in favor of computerized displays.

This thesis is a follow-on to previous M.S. thesis research [Ref. 2] dealing with classification of terrain. That work is an effort to classify individual cells of terrain based on slope and curvature. Actual terrain data from the Fort Hunter Liggett Digital Terrain

Database is utilized in an initial attempt at this sort of work on realistic, dense, and accurate elevation data. It succeeds in its basic attempt to classify terrain, but leaves several new avenues to explore in improving the algorithms.

The major emphasis of this work is an attempt to expand and improve upon the previous work utilizing a graphics tool, B-splines (a type of bicubic spline), to smooth the elevation data in an effort to remove inconsistencies in the data and facilitate recognition of terrain features. Additional algorithms are programmed to display the available data in a variety of formats to assist in terrain analysis and recognition.

B. ORGANIZATION OF CHAPTERS

Chapter I gives a brief description of the organization of this thesis and an introduction to the work to be accomplished.

Chapter II reviews some previous work in the area of terrain classification. Several investigators have developed algorithms that classify terrain based on slope and curvature [Ref. 3, 4]. These initial works are the basis for additional work on test elevation data in an effort to analyze individual terrain cells [Ref. 5]. This effort is further expanded and formalized utilizing state of the art hardware and software for actual elevation data from the Fort Hunter Liggett Digital Terrain Database [Ref. 2], in an effort to improve speed and efficiency.

Chapter III is a statement of the problem. It examines the database being utilized and introduces the concept of B-splines to represent surface patches of terrain data. Also included is a description of the hardware environment in which the system will function.

Chapter IV is a detailed look at the algorithms that are developed during the research. The first section is a discussion of modifications to previous algorithms. Three new files of algorithms are developed to handle contours, slopes, and B-spline smoothing. Additionally, the drawing algorithms are significantly expanded to increase

flexibility in the displaying of the results. A user's guide is included in this chapter to assist in the running of the system for the application of the algorithms to actual data.

Chapter V is a review and analysis of the output. It discusses the types of output that the user can expect and how to display this information in a variety of formats.

Chapter VI is the analysis of the completed work which attempts to put the results of the work into perspective and open up areas for future research.

Appendices A-G contain the source code for the project. Each of the appendices includes a different section of the algorithms developed. The algorithms are modularized in individual files to enhance modifications and improve reading and comprehension of the myriad of small functions which is a characteristic of LISP code.

C. PURPOSE

The intention of this thesis is to formalize a method for storage, manipulation, and display of realistic, dense, and accurate elevation data. The main intent is to improve upon previous work in classifying individual cells of terrain based on their slope and curvature. This type of classification can assist in identifying sections of terrain (watersheds) which have similar characteristics. The B-spline surface fitting method is utilized to smooth the elevation data in an attempt to improve classification and terrain segmentation. The ability to segment terrain into areas with common characteristics will ultimately be utilized for route planning and in the recognition of terrain by land vehicles. This is a first step in teaching a machine that is controlling a vehicle to think and act like a human driver acts under the same set of conditions and circumstances.

II. SURVEY OF PREVIOUS WORK

A. INTRODUCTION

The problem of terrain classification has existed for many years, but computer automation of this process is a relatively new and growing area of research. Proper classification of terrain by computers represents a major step forward towards teaching a machine to analyze elevation data in a manner similar to humans. This type of classification is not easy and a human, in making his analysis, utilizes many variables and heuristics when finally classifying a section of terrain. A computer has a much more difficult time when faced with a wide variety of parameters, but can handle a much larger volume once it has been properly programmed. There are many ways in which the information which is generated by a classification algorithm can be put to use. It can be used for analysis of a particular region to determine if it is suitable for development. Geological information can be used to determine if an area is suitable for exploration. Each of these are civilian applications.

There are just as many military applications which would benefit from this same classification information. In this thesis, the ultimate goal is to assist an autonomous vehicle in both route planning and terrain understanding as it traverses selected terrain. One of the main problems in this area is that the work of classification has been so computationally demanding.

A number of works in recent years attempt to identify key terrain features using a variety of methods [Ref. 2-5]. This chapter reviews a few of these which have significance to the goal of this thesis. Additionally, several methods of surface representation are reviewed.

B. SURFACE CLASSIFICATION

A number of works were reviewed as a basis for the algorithms which are utilized in this thesis. The development of the algorithms has been an evolutionary process with each work providing key concepts and information of value to subsequent efforts. Several of the works reviewed here make significant inroads into developing definitions for proper classification of terrain and consequently are of prime concern.

1. Basic Concepts

A recent work details several concepts essential to the development of an algorithm for terrain classification [Ref. 4]. The work is an effort to find a data structure to store topographic and other surface information in a much more computationally efficient manner. The structure selected should also be capable of allowing easy reproduction of the original surface. A review of some of the concepts and conclusions provides a key background for the work of this thesis.

a. Surface Storage Methods

The first step in developing a data model is to determine a method of data storage. Three methods of maintaining surface data are detailed [Ref. 4]. The discussion of surface storage methods is significant since it validates the later choice of a grid elevation database as the one best suited for a realistic system where a wide variety of terrain is encountered.

The *grid digital elevation model* is the first method of surface storage considered. Elevation data is maintained at points determined by a regular grid of evenly spaced points. This method is inefficient in large flat areas as the data being stored is redundant. However, it has the advantage that it requires a smaller amount of storage and is computationally efficient in areas of typical relief (areas where there are a number of terrain features and a variety of elevation values). Since the elevation data is typically placed in a matrix data structure, there is no need to store the x and y coordinates as these

can be quickly determined mathematically from the storage location. This method also gives almost instantaneous access to any location in the grid.

Another method discussed is the *contour digital elevation model [Ref. 4]*. Elevation data is stored by utilizing line traces of the contours. The elevation points are typically maintained in a linked list structure with pointers to successive elements. To access a particular point in the list or to find a neighbor to a particular point can be extremely slow and cumbersome. This method is generally good in flat areas as a minimum of space is required to store data covering large areas of terrain. However, in areas of typical relief, it is no more efficient than the grid digital elevation model and is much more computationally intensive.

The *triangular irregular network model* is yet another approach to surface representation [Ref. 4]. Elevation points are stored in the form of a triangular tessellation, where all of the terrain cells within a triangle have common surface characteristics. This method again has definite storage advantages in relatively large flat areas, but fails to provide any advantage in areas of typical relief. In the later areas, a large number of triangles are required. Additionally, computational speed is poor in moving within any area. Finally, this system causes a major problem with artifacts when transitioning from relatively flat areas to areas with a much greater amount of relief. In the case of a plain cut by a deep ravine, within the ravine, the number of triangles is large, while on the plain, they are far apart. At the junction of these two types of terrain, there are a number of elongated triangles which result from the transition from the large triangles on the plain to the more closely packed smaller triangles in the ravine. These elongated triangles are a result of the storage method and are not true representations of the existing elevation data.

b. Terrain Types

Key to any classification of terrain is the need to understand the possible features which might be encountered. Certain types of terrain are information rich and contain a wealth of useful facts when compared with the typical terrain cell. The ability to properly define all of the terrain features in a format which can be understood by the computer is essential to developing any sort of computerized system of terrain recognition and classification.

Peaks, pits, saddles and **passes** are key pieces of terrain in that they are the points of juncture for the other type of common terrain features, **ridges** and **ravines**. For each type of terrain, essential bits of information are known. For example, a **peak** is a point of local maximum. No point in the immediate vicinity has higher elevation. Additionally, the slope at the **peak** is zero. Similar information defines the other features. This type of information is important, but of even more significance to this thesis is the development of **ridge** lines and **ravine** lines. **Ridges** and **ravines** are defined in this work as regions of high convexity/concavity. Identifying these features is essential to locating regions of terrain with similar characteristics. The concept of a **ridge** line is closely linked with a slope line because they both cross at right angles to contour lines. **Ridges** are the boundaries between different areas of terrain that can be called basins, while **ravines** are the boundaries between hills. This information is critical to developing an algorithm to classify sections of terrain.

c. Locating Hills and Basins

With the simple definitions just mentioned, algorithms to locate **ridges** and **ravines** are considered [Ref. 4]. The first step discussed is the location of hills and basins. The trace of the line separating different hills and basins are actually the **ridge** and **ravine** lines which are ultimately being sought. A number of different methods are outlined [Ref. 4] and are summarized here.

(1) Sorting. The first method of locating hills and basins utilizes a simple sorting of all the data points based on elevation. Starting with the lowest point, each point is checked to see if it is a neighbor to a point already flagged as being in a particular basin. If so, it is flagged with the same basin identification as the previous basin. If not, a new basin identification is started. This method is investigated [Ref. 4] and begins by looking at four orthogonal neighbors of a given point. In an effort to improve recognition, it expands the search to the eight nearest neighbors, and eventually utilizes twenty neighbors. However, it is not successful as it can never overcome a problem with the sorting algorithm not being able to place neighbors with equal elevations in the same basin next to each other in the sorted list.

(2) Hill Climbing. The second method discussed [Ref. 4] pushes neighboring points of an initial location onto a stack. Points are popped off in a last-in-first-out order checking again to see if there is a neighbor lower than itself already in a flagged basin. This algorithm works only until it reaches a saddle point where it allows the growth of the basin to pass through the saddle and wrap around to the other side of the hill.

(3) Use of Slope Lines. The first two methods considered here are not successful because they fail to follow the rules for defining slope lines and instead rely solely on elevation. The pure definition of a slope line is a line of greatest inclination through a point. Additionally, it is perpendicular to contour lines. This method divides the cell in question with two diagonals effectively cutting it into four triangles. An elevation for the midpoint of the cell is calculated from the mean of the four corners. Equations for each of the triangles are developed. With this information, slope lines are determined and traced upward to the **peaks**. Once this has been completed, it is necessary to trace the boundaries between adjacent hills. These boundaries are the **ridge**

and **ravine** lines for the terrain. This method appears to be successful, but is very expensive computationally.

(4) Local Procedures. The final method discussed [Ref. 4] involves utilizing procedures based on characteristics of terrain immediately adjacent to the cell in question. Several methods are considered. One method is based on plotting the projection of the area immediately around a point onto a plane and working with that information to determine the classification of the cell. A second method requires taking cross sections of the terrain in the immediate vicinity and searching for curvature. It is a basic modification of this method which is ultimately utilized in this thesis for classification. The third method involves a paradox of the slope definition. A **ridge** line can not be traced by following the steepest slope. A **ridge** line is a special case of a gradient line possessing the property of unstable equilibrium. That is, water does not run down a **ridge** line but falls off on either side onto another gradient line. The water does, however, flow into a **ravine** at the bottom of the hill defined by this **ravine** and the **ridge** above. Since the concept involved in this paradox is difficult to program, the third local method developed [Ref. 4] utilizes a system of marking what is not being sought. It makes a pass through the terrain flagging the lowest point in a cell. This can obviously not be part of a **ridge**. On completion of a pass through the area every place is flagged except for **ridge** lines.

d. Thinning of Results

Each of the methods described above produces results, and yet none is in the form of actual **ridge** and **ravine** lines, the ultimate goal. The output from these methods is **ridges** and **ravines** with some noise or extra cell classifications referred to as *clouds*. It is necessary to somehow thin out the information which is produced. These clouds can be thinned by making pixel by pixel comparisons to determine which pixel more closely depicts the line anticipated. This process can be repeated until only a

skeleton remains. The areas remaining inside these skeletons are areas with similar characteristics. One difficulty encountered occurs when two pixels have similar characteristics. Care must be taken to ensure that both are not thinned. Additionally, end pixels are often eliminated. If several passes are required in thinning, an end line can be significantly shortened and the potential for closure of lines is reduced. Since the ultimate goal of all these efforts is to locate **ridge** and **ravine** lines to segment similar terrain, problems associated with elimination of line pixels must be minimized.

e. Junction Points

As stated earlier, **peaks, pits, saddles,** and **passes** are junction points which connect the various **ridge** and **ravine** lines to segment the terrain. In making the connection, care must be exerted to ensure that the algorithm does not loop around a particular region. It does not matter if the search for paths goes clockwise or counterclockwise, as long as the junction points are properly handled so that the algorithm chooses a new exit route each time it visits a junction point.

f. Model Creation

The author concludes this work [Ref. 4] with a discussion of an implementation of the algorithms for storage and classification detailed above. The elevation data is read into a storage structure. The concave/convex cross section method and the highest/lowest corner method are applied to the raw data creating a bitmap which is subsequently thinned to develop a network of ridge and ravine lines. These lines are displayed for comparison.

g. Results of Work

Various methods of representing a topographic structure are discussed in this basic analysis of terrain concepts [Ref. 4]. While several of the methods are flawed, others are implemented and compared for realism. None of the methods appears to work consistently, but they each achieve a certain level of success. Some of the ideas and

concepts discussed in this work are important to understanding the problems and difficulties encountered in properly classifying terrain. The next work reviewed provides a more mathematical formulation of the surface structure definitions.

2. Cubic Approximation to Terrain

While the previous work contains some basic definitions of the types of surfaces to be encountered and classified, another work [Ref. 3] attempts to codify these types mathematically. The main goal of this work is an attempt to describe a sketch of the gray line intensity of a digital image. To accomplish this, the author tries to first develop a robust representation method to discover and classify variations in surfaces. This surface representation is the basis for a "Primal Sketch" for computer vision. This sketch is able to scan, analyze, and redisplay digital images with a minimum of roughness.

a. Basic Model

The model utilized to represent the surface is a bivariate cubic [Ref. 3]. The neighborhood surrounding each pixel is fitted with this mathematical representation, thus allowing calculation of the first and second derivatives. The computation of these derivatives is important as they are the basis for the appropriate classifications. The first derivative yields the slope and the second derivative yields the curvature. In this work, every combination of values is exhaustively considered. Some of the combinations yield the same classification, or a generalized classification, but the important distinctions stand out as easily recognized features. What is of prime concern is that a mathematical model, easily understood by a computer, is developed. From this model algorithms can be developed to utilize this information for identifying key terrain features.

b. Mathematical Basis

The basic concept applied in this work [Ref. 3] is that the surface can be represented by mathematical equations which readily result in computation of both the

first and second derivatives. With this information, every combination of associated data is checked for the appropriate classification. As with the previous work, some of the basic types of terrain encountered are the **peak, pit, ridge, ravine, saddle, flat,** and **hillside**. Several additional subclassifications are developed for the hillside. These additional classifications cover most of the more generalized cases where the maximum or minimum conditions do not occur. The basis for classification is five values derived from the mathematical formulas for the local surface patch. These values are for the slope, curvature along the major and minor axes, and gradient vectors. Figure 2.1 shows some of the possible types of terrain which can be encountered. The terrain feature at the top is a **peak** with a well defined **ridge** running on an east-west direction. At the bottom of the figure is a terrain feature containing a **ravine**. A **ravine** has positive curvature along the major axis while curvature along the minor axis is below a threshold. The point in the **ravine** between the two **ridges** is a **saddle** or a **pass**. The difference is subtle and is determined by the sign of the eigenvalue of larger magnitude [Ref. 5].

c. Mathematical Values

The values required for classification are a result of computations which the computer is imminently qualified to perform. Given a function at a point *f(x,y)* which defines the surface, the slope at the point is determined by

$$\left(\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2 \right)^{1/2}. \tag{2.1}$$

The values for the curvature can be determined from the second derivatives contained in the *Hessian* matrix defined as

$$H = \begin{bmatrix} \partial^2 f / \partial x^2 & \partial^2 f / \partial x \, \partial y \\ \partial f / \partial y \, \delta x & \partial^2 f / \partial y^2 \end{bmatrix}. \tag{2.2}$$

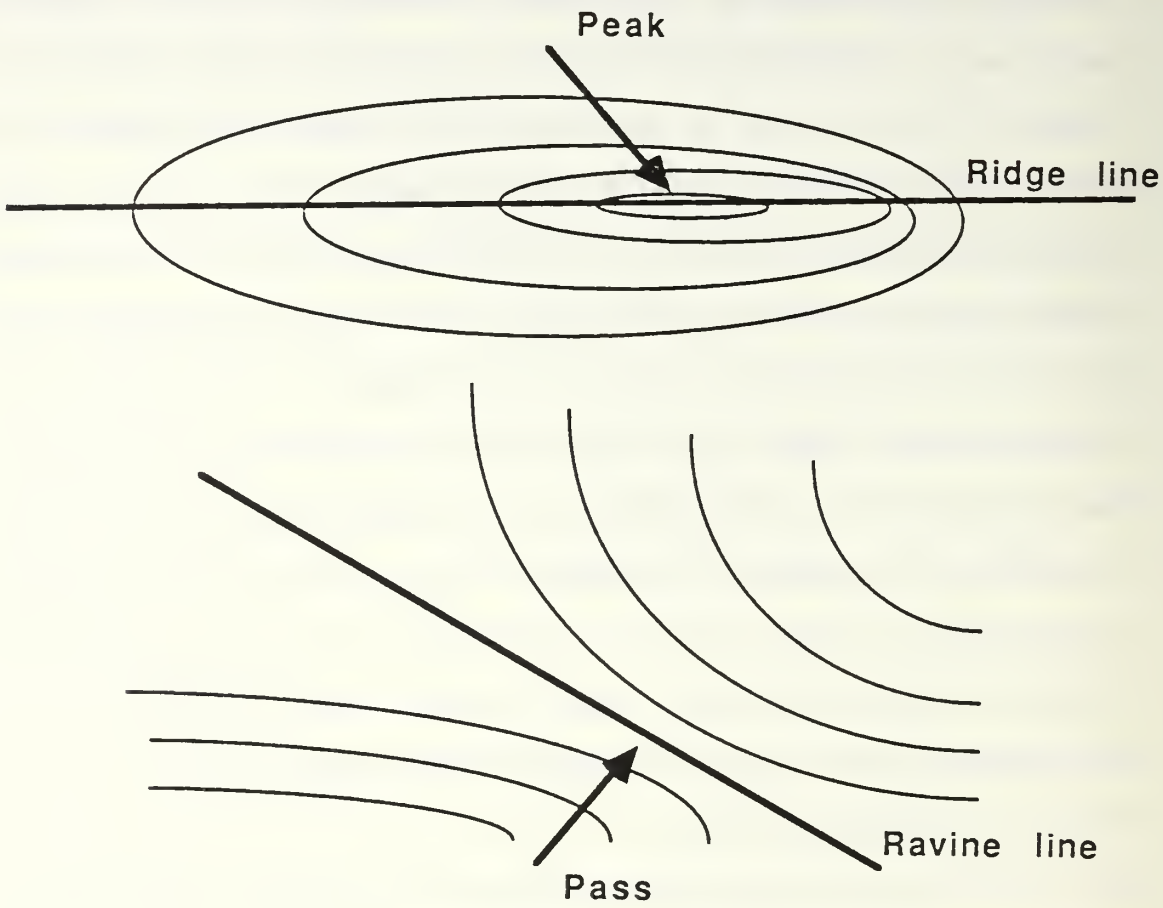

Figure 2.1 Sample Terrain

Equations 2.1 and 2.2 form the basis for later works which attempt to classify terrain based on the slope and curvature of an individual cell. From them, all the essential information can be developed. A derivation of these values is not included as it is not necessary for understanding and can be found in the literature [Ref. 3].

3. Classification Based on Slope and Curvature

More recent work [Ref. 5] utilizes the types of terrain and mathematical values just discussed to classify individual sections of terrain using a quadratic approximation of

the surface based on least squares fit. This is a specific version of a generalized quadratic equation and is of the form

$$f(x,y) = k_1 + k_2x + k_3y + k_4x^2 + k_5xy + k_6y^2. \tag{2.3}$$

Equation 2.3 provides a representation of the center pixel of the surface and the eight surrounding neighbors. As with work utilizing bivariate cubic equations [Ref. 3], values for the slope and curvature are easily derived from this formula. With the simplified quadratic equation, the slope is defined as [Ref. 5]

$$(k_2^2 + k_3^2)^{1/2}. \tag{2.4}$$

The Hessian matrix for this equation is evidently

$$H = \begin{bmatrix} 2k_4 & k_5 \\ k_5 & 2k_6 \end{bmatrix} \tag{2.5}$$

from which the two eigenvalues, λ_1 and λ_2, can be obtained [Ref. 5]. In both of the equations above, the values for the various coefficients are determined by a matrix manipulation of a basis matrix and an elevation matrix. Terrain data can be classified utilizing the sign of the eigenvalues determined from Equation 2.5. Subsequent work to modify the algorithm to analyze dense accurate elevation data also has been conducted [Ref. 2].

A smaller version of the possible classification types previously developed [Ref. 3] was subsequently utilized for classification based solely on curvature [Ref. 5]. This matrix, which is based on the sign of the eigenvalues of the Hessian matrix, is included as Table 2.1. This Table represents the various classifications which can be derived based on curvature. It is important to note that two types of classification are impossible. If the sign of the maximum curvature is zero, it is

TABLE 2.1

CLASSIFICATION OF TERRAIN CELLS
BY EIGENVALUES OF HESSIAN MATRIX

λ_1 = eigenvalue of larger magnitude
λ_2 = eigenvalue of smaller magnitude

sign of λ_2 \ sign of λ_1	+	0	-
+	pit	impossible	saddle
0	ravine	planar	ridge
-	pass	impossible	peak

impossible for the smaller value to have any value other than zero also. Thus, from the three x three matrix there are only seven possible classifications.

C. TERRAIN CLASSIFICATION

A second work in the area of terrain classification expands on the previous effort by applying the algorithms to actual terrain data and not just test data [Ref. 2]. Work using slope and curvature information focuses on transferring the algorithms from the ISI machines and installing them on the Lisp machines which were found to be much faster [Ref. 2]. There are some modifications that need to be made to this work to allow completely accurate representation of the classification types. These changes are contained in Chapter 3.

The algorithms developed [Ref. 2] contain two methods for terrain classification. The first is based on the curvature and utilizes the possible classification types discussed above. The algorithm utilizes a **primary** and **secondary** classification scheme. The **primary** classification is based on the value of the slope and can be either flat, safe, or unsafe. Only if the **primary** classification is safe or unsafe will the algorithm determine a **secondary** classification. The **secondary** classification is based on the curvature of the cell and can be one of the seven possible classifications detailed in Table 2.1.

It is important to classify **ridge** and **ravine** lines since they segment the terrain into regions with common characteristics. The **peaks**, **pits**, **passes**, and **saddles** are mainly utilized for closure to connect two **ridges** ascending a hill, or conversely, two **ravines** descending to either side of a **pass** or **saddle**. It is important to note that this method does not locate actual **ridge** lines.

A third method of terrain classification utilizing eigenvectors is introduced to augment the classification based solely on curvature [Ref. 2]. An *r-line* cell is defined as one in which the direction of maximum curvature (principal eigenvector) is orthogonal to the direction of maximum slope (gradient). If the principal (largest) eigenvector is negative, the r-line is a **ridge** line. If it is positive, the r-line is a **ravine** line. **Ridge** lines and **ravine** lines thus identified are extremely important in terrain classification and recognition because every hillside, also referred to as a *watershed,* is bounded below by a **ravine** line and above by a **ridge** line. Once a machine is capable of making this distinction, it has taken a significant step towards analyzing terrain as a human does.

Because the orthogonality method of determining **ridge** and **ravine** lines is considered to be the more important method of classification, it becomes the priority for display. The algorithm first checks to see if a cell is classified as a **ridge** or **ravine** line. If so, this information is displayed. If the algorithm is unable to make a classification

with this method, the cell is next checked for **primary** and **secondary** classifications based on slope and curvature to determine the value for the cell.

D. TERRAIN SMOOTHING

The work previously completed [Ref. 2] succeeds in classifying individual cells of terrain, but the final product contains a large amount of unnecessary or unwanted details. Some of this is corrected with modifications to the algorithms, but there is still a *speckling* effect. This includes ridges that contain gaps and individual cells in the middle of open areas which are classified with no association to the surrounding cells. As stated earlier, one of the main goals of the present work is to eliminate this speckling effect.

1. Spatial Filtering

One possible method for terrain smoothing is spatial filtering. This involves a Fourier transform approach to filtering out unwanted pieces of data by combining pixel data with that of surrounding pixels [Ref. 6]. With this method, unwanted images or classifications can be eliminated. It is not pursued since there is still additional work which can be done with polynomial smoothing in an effort to achieve the same effect. In addition, the spatial filtering technique is a much more computationally intensive method.

2. Planar Patches

Another method currently being explored is the use of planar patches to represent the elevation data. Planar patches are regions of terrain which lie in an area that possesses similar characteristics. This method determines a value for the magnitude and direction of the slope and utilizes this information to group cells within similar regions. Cells that fall within specified thresholds are linked together for future

reference. This method is useful in locating regions with similar characteristics for route planning algorithms. [Ref. 7]

3. Graphics Representations

A final method considered for smoothing the elevation data is the use of computer graphics techniques to represent the data. In graphics, a number of methods are available for depicting irregular surfaces. Each method has advantages and disadvantages which can appeal to various users. In attempting to improve the analysis of terrain cells, it is intended to first smooth the terrain to possibly eliminate minor errors and imperfections in the raw elevation data. To do this, some method must be utilized to represent the surface for the purpose of smoothing. While there are many methods available, three of the more common methods are reviewed here. Each method employs a form of curve representation expanded to three dimensions.

In two-dimensional curve representation, there are typically four control points which determine the shape of the curve. Depending on the method chosen, the curve can pass through the first and last control point, the two middle control points, or none of the points. Depending on the purpose of the representation, the appropriate method is determined. In expanding this concept to three dimensions for surface representation, a four by four grid of control points is used. In most graphics texts, the following three types of surface patches are commonly found.

a. Bezier Patches

The Bezier patch has significant differences from the other two methods of surface patch representation discussed here. In this method, the surface actually covers the entire area bounded by the sixteen control points. In the other two methods, the patch only covers the surface defined by the four inner control points with the outer control points controlling the shape of the surface.

With the Bezier patch, the surface passes through the eight control points on two opposite edges. The eight other points help determine the shape of the surface. This method is convenient for quickly determining large sections of a surface but, because of the mathematics involved, has no continuity at adjacent edges. Also, if in a large array of control points the surface being defined is shifted just one increment in either direction, the overlapping area of the surface patch will not be exactly the same as the previous definition. This is not significant for some applications, but in an algorithm which is striving to achieve maximum accuracy of surface representation the variances encountered are considered to be too large to ignore. It is just this type of minor variation in the surface of the terrain which is to be eliminated in this thesis.

b. Cardinal Patches

The Cardinal patch method, like the B-spline method to follow, only depicts a patch within the inner four control points. With the Cardinal patch, the surface passes through these four interior control points. This method has definite advantages over the Bezier patch method. However, the raw elevation value at the control points is the piece of data that is to be revised in an effort to smooth the surface. This method, which has the surface passing through these same four control points does not alter the raw elevation value, but returns the exact same value at these control points. It is therefore also unsuitable for the work of this thesis.

c. B-spline Patches

The B-spline patch does not require the surface to pass through any of the control points. The advantage of the B-spline patch is that it is a cubic representation which maintains both first and second derivative continuity at the edges. This ensures that classification of adjacent cells using slope and curvature information is accurate and consistent. This method of surface representation can be used to shift the present surface representation across the entire data grid and the surface will be a smooth representation

of the elevation data. Extracting the elevation data at the appropriate control point yields a smoothed elevation value which can subsequently be input to the classification algorithm.

E. ROUTE PLANNING

Another area of research relevant to this thesis uses classification algorithms to determine if sections of terrain can be detected with similar qualities [Ref. 8]. This is important in that if a particular section of terrain can be determined to contain a number of cells which have equal classifications, then these sections can be grouped into even larger pieces of terrain that can be given a similar code for traversal purposes. This becomes important in route planning because, rather than considering each cell individually, it is possible to consider whole sections of terrain with similar qualities.

As an example, all the terrain between a **ridge** line and a **ravine** line is in an area that is called a watershed. This is an area in which all the water in this one section flows in basically the same direction. By determining a number of watersheds across a region, whole areas of terrain can be divided into sections with similar characteristics. This significantly reduces the computational requirements necessary for a route planning algorithm, as once a traversal cost value is calculated for a region, it does not have to be recalculated as long as the vehicle path remains in that region.

F. SUMMARY

This chapter reviews a number of previous works which utilize a variety of algorithms relevant to terrain classification. Also included is a comparison of several graphics methods for representing terrain surfaces. Each of these works has contributed to a better understanding and appreciation of the complexities involved in teaching a machine to classify individual terrain cells. The fact that none of the works discussed are able to achieve a fully working method for accurate classification demonstrates the

difficult nature of the problem. With each iteration of the process, some progress has been made. The ultimate goal is a machine algorithm capable of detecting various surface attributes.

The next chapter discusses the problems associated with the terrain classification method utilized by this thesis. A review of the Fort Hunter Liggett Digital Terrain Database details the idiosyncrasies of that system. The B-spline surface representation is presented as the chosen method for smoothing the raw elevation data.

III. DETAILED PROBLEM STATEMENT

A. INTRODUCTION

There has been much work accomplished in the past few years in the area of classifying terrain based on the slope and curvature associated with a particular grid cell. While this has been successful in a number of ways and it has been possible to determine much about individual grid squares from these methods, attempts to segment terrain into objects with consistent properties have been less successful. This results in part from the problem of the terrain representation method not being a smooth representation of the data. This can be caused by one of two things. Either the data input to the system is not accurate, or the mathematical algorithms which are being used to simulate the data are not accurately depicting the data.

If the data being utilized is not accurate, it is obvious that there will be problems with classification. All of the elevations in the database used in this thesis were input by hand and are in whole feet. A slight variance in the data can cause the algorithm employed to recognize a curvature in the terrain that is not actually present. A difference of only a couple of feet can cause a depression to appear where in fact the terrain is flat. While these types of errors might not affect a large number of cells in a grid square of 6400 cells, they might exist in sufficient numbers to preclude the algorithms from recognizing **ridge** and **ravine** lines that are connected and thus identify a watershed.

Likewise, the algorithm might not be adequate to properly depict the data, thus yielding results that are not optimal. In previous work [Ref. 2], a local quadratic equation is utilized to represent the data surrounding each pixel. With this type of equation, there is no second derivative continuity. This implies that there is no guarantee

of continuity at the edges. Without this type of continuity, there are problems encountered with adjacent cells not being properly classified.

This chapter discusses both the database and the mathematical algorithms utilized to represent the raw data. This thesis expands upon previous work to solve some of the problems associated with terrain classification and recognition. It is hoped that some of the problems encountered in previous efforts can be resolved while opening up new areas for future research.

B. HUNTER LIGGETT DATA

The Fort Hunter Liggett Digital Terrain Database is a condensed representation of the topography of the entire military complex which is just west of Highway 101 in Southern California. The database contains all of the standard UTM grid squares in the region. The terrain covered is excellent for study as it encompasses such a wide variety of features. In some areas large flat spaces with a minimum of variance are encountered. Less than a kilometer away steep hills exist which rise over 1000 feet above the plains below. Throughout the area there are a substantial number of features which are perfectly suited for classification.

1. Resolution of Data

In extracting information from the database, the user can select one of two resolutions, low or high. The actual data in the database contains an elevation/vegetation value for every 12.5 meters. This is considered to be the high resolution. Additionally, low resolution data can be extracted which yields information on intervals of 100 meters. When creating data files to work with, it is also possible to extract the data in a size suitable to the needs of the project. A 1km x 1km grid, a 2km x 2km grid, or any size up to a 10km x 10km grid can be selected. Within this region, the data can be extracted in the high or low resolution format. In this work, the high resolution data is utilized to

provide better terrain clarification in an effort to more accurately classify each of the grid cells.

2. Accuracy of Data in Database

The Fort Hunter Liggett Digital Terrain Database was provided by the Army and has proved to be an extremely useful tool in the work of terrain classification. The database has been created by hand from the standard topographic maps and represents an enormous amount of work. The magnitude of the effort can be appreciated only by considering that the database contains both elevation and vegetation data for points in a grid every twelve and a half meters for an area of over 1200 square kilometers. While working with the database, it is apparent that the data is highly accurate although there are two minor inconsistencies in the database that need to be understood to allow for proper interpretation of results.

a. Hilltop Smoothing

Often, when the top of a hill or ridge occurs near a contour line where several cells are just above the local contour, these elevations are often all identical to the contour line, thus yielding a flat hilltop or ridge. While it is quite possible that this is in fact the case, practical experience has shown that there are very few hills that suddenly go flat right at the contour line. In reading and analyzing the output from the system it should be remembered that there is a possible minor error in the elevations of tops of hills and ridges.

b. Contour Crossing in Flat Terrain

There is another minor inconsistency encountered when handling the data. When a cell crosses a contour line, that cell has the correct elevation but the elevation on either side of the contour line is sometimes less than the contour line. This erroneously indicates some form of lip or small ridge which follows the contour line. This occurs mostly when there is a finger of land reaching out along a relatively flat area and the data

has gone up and then dropped back down. When the data is consistently rising or falling, there is generally not a problem. Of course this is only a minor problem and can be corrected later with an expert system that allows the user to change or correct data in the database. At the present time, these inconsistencies in the data only cause some minor problems when displaying the contours and are mostly eliminated by the B-spline smoothing algorithms.

C. B-SPLINE SMOOTHING

In an effort to properly identify each cell, it was determined that the B-spline surface patch is the best suited for the anticipated results. This form of surface representation was picked because both first and second derivative continuity is maintained at the connecting edges. It was hoped that this continuity at the edges would provide a smoother surface representation and eliminate problems with adjacent cells not being properly classified. By using the B-spline patch to represent the surface, and extracting a "smoothed" surface elevation, a better classification of cells can be achieved which improves the ability of the system to determine regions with similar characteristics.

The derivation of the B-spline algorithms can be found in several texts on graphics which are commonly available [Ref. 9-12]. These works contain sufficient explanation of the theory of splines for those interested in a more thorough understanding of the concept. This thesis does not attempt to expand upon the derivation of B-splines, but rather employs them for practical purposes.

As with each of the surface patch methods mentioned in Chapter 2, the B-spline surface patch utilizes a basis matrix. This matrix is developed by solving simultaneous equations for the cubic representation for the surface and the end conditions for the control points. The basis matrix for the B-spline method [Ref. 13] is established as

$$M_b = \begin{bmatrix} -1/6 & 3/6 & -3/6 & 1/6 \\ 3/6 & -6/6 & 3/6 & 0 \\ -3/6 & 0 & 3/6 & 0 \\ 1/6 & 4/6 & 1/6 & 0 \end{bmatrix}. \tag{3.1}$$

The formulas for representing the B-spline surface are taken from a recent technical report on surface representation [Ref. 14]. The B-spline surface representation is developed from several matrix manipulations utilizing the basis matrix in the equation

$$z(s,t) = S\ M_b\ Q_z\ M_b^T\ T^T \tag{3.2}$$

where M_b is the basis matrix from equation 3.1 and M_b^T is the transpose of the basis matrix. The matrix Q_z contains the z-coordinates (elevation) for the control points of the surface in matrix format

$$Q_z = \begin{bmatrix} P_{C1} & P_{C2} & P_{C3} & P_{C4} \\ P_{C5} & P_{C6} & P_{C7} & P_{C8} \\ P_{C9} & P_{C10} & P_{C11} & P_{C12} \\ P_{C13} & P_{C14} & P_{C15} & P_{C16} \end{bmatrix}. \tag{3.3}$$

The 16 control points are depicted in Figure 3.1. This z coordinate is the only direction in which the values are changing, since the x and y coordinates are fixed by the terrain grid, and consequently the only direction which affects the surface patch. Finally

$$S = \begin{bmatrix} s^3 & s^2 & s & 1 \end{bmatrix}, \tag{3.4}$$

and

$$T = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix}. \tag{3.5}$$

These two matrices represent the location on the surface patch at which the elevation is desired. The value for s and t can be varied from zero to one and substituted into each of

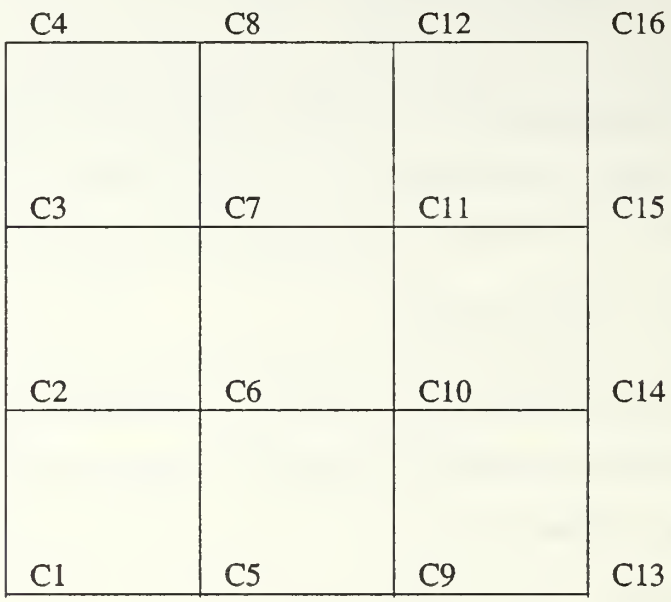

Figure 3.1 B-Spline Control Points

these equations. The system performs the necessary matrix multiplications to extract the smoothed elevation for any point on the surface patch.

In this thesis, the only values which are utilized for s and t are zero and one. These values can be be selected in any combination to produce the elevation at each of the four corners of the surface patch. This is adequate to extract the smoothed elevation at any of the interior control points. Thus, by utilizing the basis matrix and the z-geometry matrix, a mathematical representation for the surface represented by the center square of the grid in Figure 3.1 is developed. From this surface a smoothed elevation for the appropriate control point can be extracted.

D. HARDWARE ENVIRONMENT

All of the work in this thesis was completed on a Symbolics 3675, utilizing Common Lisp. This machine has a color monitor attached for displaying the various computational results. The machine is not as fast as would be expected of a real-time system, but is significantly faster than the machine which was previously utilized [Ref. 2]. The machine is networked with three other Lisp machines for which it acts as the file server. This has some effects on computational speed, but the intention of this thesis is not to compare or contrast computational speeds. The system is also connected to the Unix system in the Computer Science Department which allows for backup and storage of the large volume of programs and data generated during this research.

The color monitor attached to the Symbolics machine can operate in either eight or 24 bit mode. Originally, the eight bit mode was used. This was the faster mode because it required less calculation for displaying results. However, as more color ramps were generated, it was difficult to achieve the proper color contrast in eight bit mode and a switch was made to the 24 bit mode. This was an easy tradeoff as speed of calculation was not essential.

As a side note, the Symbolics machine operates normally in interpreter mode. Regions or buffers of code can be compiled which significantly improve the speed of the algorithms. For any follow on work in which speed is important, all modules should be compiled prior to execution.

E. SUMMARY

The major work of this thesis is to expand and enhance previously developed algorithms for the classification and display of data. The database being utilized is a realistic system which contains dense, accurate elevation data. The algorithm appears to

be unable to fully describe areas of terrain with common characteristics. It is hoped that the use of B-splines to smooth the raw data will enhance recognition of terrain cells. This capability is a major step forward towards a goal of teaching a machine to make the same sort of terrain analysis of which a human is capable. However, in the case of the machine, its increased memory and computational capacity will significantly overwhelm the human capacity. The next chapter details the development of the algorithms, including enhancements to previous work as well as new algorithms developed to implement the B-spline smoothing method.

IV. ALGORITHMS DEVELOPED

A. INTRODUCTION

Previous research in the area of terrain classification has resulted in a number of useful algorithms being developed which were available at the start of this work [Ref. 2]. This thesis modifies and improves those algorithms in addition to developing several new algorithms to assist in the classification of individual terrain cells. Each set of algorithms is maintained in a separate file to improve readability and enhance modularity. While each file is independent of the other main files, there are two files containing global constants and functions which are utilized by all of the algorithms.

B. TERRAIN CLASSIFICATION ALGORITHMS

As already noted, this work is a follow on thesis to previous work in developing algorithms to classify individual terrain cells [Ref. 2]. The bulk of the code inherited was in the classification algorithms. During the initial research of this thesis, several inaccuracies in the existing algorithms were discovered that required correction or improvement before additional work could proceed.

1. Normalization of Data

On review of the grid square which has been previously analyzed (fq5886) [Ref. 2], there are some areas which are displayed as **Unsafe Slope** (slope greater than 28 degrees) that appear to be relatively flat. Close review shows that the actual slope in this area is approximately 5 degrees. The algorithm was retested on a section of data that is planar and has a 10 degree slope. It was determined that the algorithm had not normalized the change in elevation in the x and y directions. Previous research in this area [Ref. 5] dealt with a unit of distance between each of the elevations.

However, this work had not run the algorithms against actual data. When the algorithms were converted, the data was not normalized for the 12.5 meters which is the unit of distance between each elevation point in the Fort Hunter Liggett data. The algorithm is modified for this normalization and produces the desired slope for the test data and likewise for the actual data.

The problem of failing to normalize the data had been masked by the fact that the original algorithm had produced results that appeared believable. Flat areas appeared where relatively flat areas were known to exist and some more steeply sloping areas appeared as unsafe. However, in some of the areas between these two maximums, there were no common areas of classification and a speckling effect of all types of classifications appeared in whole regions of the grid square. When the revised algorithm was run against the actual data for grid square fq5886, none of the grid square was classified as unsafe and there was a much improved resolution for **ridge** and **ravine** recognition along with larger areas of the grid square classified into similar types.

2. Negative Discriminant

Once the elevation data was normalized for the distance between points, a new problem surfaced. Previously, the numbers being manipulated for slope and curvature were large and when calculating the eigenvalues there was no problem. However, now that the numbers were significantly smaller, the order of magnitude was decreased and negative numbers were generated in the formula for calculating eigenvalues. These small negative numbers occurred in the discriminant of the equation for calculating the roots and resulted in imaginary roots. Mathematically, this had been proven impossible [Ref. 5], but due to roundoff of small numbers it was nonetheless occurring. Closer survey showed that when the parameters passed to the formula were approximately equal, the discriminant approached a value of zero. However, this value occasionally was less than zero. Test runs against the data showed that the error occurred

in the twelfth or thirteenth decimal place. This was significantly smaller than the amount it was subtracted from and consequently unimportant. A conditional was placed in the code to set any discriminants less than zero to zero. A test of classification showed that this modification did not change a single classification in a grid of 6400 cells. This was considered to be a minor change, but essential to getting the program to execute properly.

3. Default Classifications

During analysis of the previous work it was noticed that there were classifications occurring which were the default classifications. These should not have occurred and were erroneous. Several typographic errors were detected and corrected that allowed the algorithms to properly classify each section of terrain. Once these changes were made, the algorithm classified each cell as one of the seven possible types of terrain and no longer displayed the default classifications.

4. R-line Tests Not Shown

One of two methods used to classify cells [Ref. 2] is an orthogonality test. A check is made to determine if the direction of maximum curvature is orthogonal to the direction of maximum slope. If so, this is determined to be a ridge-line cell or ravine-line cell based on the sign for the value of the curvature. This test is conducted on each and every cell regardless of whether it is flat, safe, or unsafe. The code to display this test was actually at the end of a conditional that covered the seven possible classifications based on curvature. This in effect caused this code to not be displayed as the cell was always classified as one of the seven types because it defaulted to the seventh (flat) if it didn't have the characteristics of one of the first six. In this thesis, the conditional to display this type of information is placed above the conditionals for curvature so that this test is the first thing displayed no matter what classification is determined utilizing the second method.

5. Rewrite of Conditional Loops

In working with the original algorithms for the B-spline smoothing, it was discovered that the main algorithms were working in an inverted order. When the program requested information for x-coordinate and y-coordinate, it actually returned mapsize minus x-coordinate and mapsize minus y-coordinate. This is a result of the data in the Hunter Liggett database being stored from the lower left corner of the grid square, working up the left side and then repeating across the grid square. When the data has been loaded into an array for manipulation, element 0,0 is in the upper left corner instead of the lower left corner. In an effort to make it easier for all to understand, the algorithms are rewritten to conform to the same method as utilized in loading the main array.

C. CONTOUR ALGORITHMS

1. Contour Intervals

The contour algorithms in this work display the terrain according to a color ramp that shows the changes in elevation according to the 40 foot contour interval of the typical topographical map grid square. These displays are surprisingly accurate when compared to the topographical map with only a few minor problems encountered. The algorithm puts a code out to a file for the elevation in each cell. This is to save time later. If the information is to be redisplayed, the computations do not have to be repeated.

The colors are ramped between 1200 feet and 1800 feet. This is the typical range of elevations in the area of this study. All elevations above and below these ranges show as the same. It would be a simple matter to expand this algorithm for a range from 0 to any desired elevation. However, since the grid squares utilized are within the ranges above, these are the ranges coded into the system.

2. Vegetation

The displaying of vegetation is purely an outgrowth of initial thesis research. At that time, a number of research areas were being explored. One possibility was an attempt to develop a full topographical map from the digital database. Part of this effort would be to extract the vegetation data from the database and display that along with contour lines and known cultural features (roads, buildings, etc.). This work was not attempted but the algorithms to display the vegetation are included here as part of an overall package to display all known data for a grid square. Future research might make use of or expand on this capability.

D. B-SPLINE SMOOTHING ALGORITHMS

The major thrust of this thesis is an effort to improve the capabilities of the system to properly classify individual terrain cells. The method chosen utilizes B-splines to smooth the data to eliminate errors and minor fluctuations in raw data. The algorithms developed in this file use these splines to smooth the raw elevation data. The elevation data is loaded into an array and each elevation point in the interior of the array is smoothed based on the surrounding points.

After initially loading the data into the array, the algorithm loops through the array attempting to smooth each elevation point. It first checks to see if a point is on the edge. Since the B-spline requires 16 control points to fully identify a surface patch there must be a minimum of elevation points around a cell to be able to develop the surface approximation. If a cell is on the edge, it is not possible to perform any smoothing and the actual elevation is used as the smoothed elevation. This can cause some minor inconsistencies around the edges but is not considered a significant problem in the overall system.

If a cell is in the interior, a geometry matrix is loaded with the sixteen elevation control points (previously detailed in Chapter 3), necessary for approximating the terrain surface. The B-spline basis matrix and the geometry matrix are utilized to mathematically represent the surface. The surface thus defined is the inner square in Figure 3.1. The next step is to extract the elevation for a known point on this surface.

Any location on the surface can be found by utilizing a variable for the horizontal and vertical directions. Letting **s** represent the horizontal direction and **t** represent the vertical direction, the values for **s** and **t** can vary from zero to one. Thus, a value for **s** and **t** of (0,0) represents the lower left corner of the surface while (1,0) represents the lower right corner.

The B-spline smoothing algorithm works through the array of elevation points from the upper right corner of the grid. For the typical condition, the algorithm calculates the smoothed elevation for the upper right corner of the surface (0,1). Special handling is necessary for the bottom and right interior columns of the data array to allow sufficient control points to determine the surface. The end result is a smoothed elevation for each data point in the array. This information is subsequently written into a file of smoothed elevation data for future use.

E. SLOPE ALGORITHMS

The slope algorithms utilize the same technique as the classification algorithms. Once the array is loaded and all the calculations have been performed it is merely a matter of writing the appropriate code to a file for future display. The slope algorithm determines if the slope is flat, unsafe, edge, or safe. If it is safe, the algorithm further ramps the color to show a range between the minimum slope of 2 degrees and the maximum of 28 degrees.

F. DRAWING ALGORITHMS

The drawing algorithms have been drastically modified from the original work which basically displayed only the classification information in the center of the screen with each cell represented by a 10 pixel by 10 pixel box. The new functions display classification, slope, contour, and vegetation. Each display is parameterized to allow the user to vary both the location of the display and the number of pixels used to represent each terrain cell. This allows a great deal of flexibility as the user can now display a grid square on the entire screen or can display several grid squares on the same screen for comparison of results. Along the side of each display, a scale is drawn to show the meanings of each color utilized in that particular display. A conditional is placed in the algorithm so that this scale only appears when utilizing an actual grid square of 80 by 80. This is done so that when working with test data on a much smaller scale or large pieces of terrain greater than one kilometer on a side the display is not dwarfed by the scale. This scale is parameterized like the main displays and can be drawn anywhere on the screen based on the user's needs.

G. USER'S GUIDE

1. Creating Terrain Elevation/Vegetation Data

The Fort Hunter Liggett Digital Terrain Database is available on the Naval Postgraduate School Computer Science Department's VAX 785 Computer to anyone who has an account on this system or any other system which is connected via the ARPANET. The programs to create data files from the master file are locally written and not fully documented. Accordingly, a quick synopsis of how to extract data from the system is included here for future reference.

An individual database can be extracted for any grid square or squares within the limits of the Hunter Liggett database. The Northern limit (in five digit standard

notation) is 95000. The Southern limit is 60000. The Western limit is 41000 and the Eastern limit is 77000. Within these boundaries, elevation and vegetation data is available. When running the program, these boundaries are displayed for the user and error checking is performed to ensure requested data is within the appropriate limits. There is a minimal amount of additional error checking in the program, so extreme care should be taken when running the program to ensure that each of the requested inputs is within the specified limits and the format requested. The program runs very quickly and can be rerun as often as desired.

A database can be created for either elevation or vegetation data. There are actually two distinct programs to run. Both of the programs to extract data for selected grid-squares can be found in the directory **/work2/terrain/ross** and can be run by merely making this the working directory with the **cd (change directory)** command. Following is a brief description of how to run the program:

- First login to the Unix System with individual login and password (a user can also enter from other machines on the network with the **rlogin** command or the **telnet** command).
- Change the working directory to the one containing the programs to extract either the elevation or vegetation data - **cd /work2/terrain/ross**.
- Type in the name of the program to run to create either elevation or vegetation data **make-database-e** (or **make-database-v**). The files are executable versions of compiled C code.
- Both the elevation and vegetation programs display information about the database and prompt for input to continue or halt the program. If it is not readily apparent what the appropriate inputs should be, the program should be halted and assistance sought from one of the staff personnel in the Computer Science Department.

- If the inputs are known and an individual database is to be created, the proper response at the first prompt is **1**. The system will now prompt for the following information:

  - **X Range** (Lower southwest corner of 1km grid)

  - **Y Range** (Lower southwest corner of 1km grid)

  - **Size of test database** (From 1 to 10 depending on need. A response of 1 generates a 1km x 1km grid. A response of 2 generates a 2km x 2km grid with 4 grid squares. A response of 10 generates a 10km x 10km grid with 100 grid squares.)

  - **Resolution** (12.5 for high-resolution and 100.0 for low-resolution)

  - **Test database file name** (Working filename supplied by user. Should be something unique to this directory so as not to overwrite any other files already here)

- Following successful input of all parameters, the program extracts the requested information from the database in binary format. If only one file is being created, the user should exit the program by inputing a **0** at the appropriate prompt. It is now necessary to convert the output from binary to integer which is the appropriate format for most systems. This is accomplished with the same program for either elevation or vegetation data. Type in **read-database** at the system prompt. The program will now prompt the user for the following information:

  - **Input File** (Enter name of file created above as input to this program)

  - **Output File** (Enter name of file to temporarily store output. The naming convention utilized in this thesis is **fq5886.1-hr-e**. The first part identifies the standard UTM 1000 meter grid square. The **1** indicates the file is for a 1km grid. The **hr** indicates the data is high-resolution (every 12.5 meters). The **e** indicates it is elevation data.)

- **Size** (Total number of data points being created. For a 1km high-resolution file this would be 6400 (80x80). For a 2km high-resolution file this would be 25600 (160x160). For a 2km low-resolution file this would be 400 (20x20).)

- At this point an output file has been created containing either the elevation or vegetation data for the requested grid location. This file should be copied into the user's directory utilizing the copy command. Once it has been verified that the file has been copied and is readable, the two files created under **/work2/terrain/ross** should be deleted to keep that directory from growing too large. The user can now exit the directory and return to his own directory by typing **cd** at the system prompt.

2. Loading the System

The algorithms developed are contained in seven files on the Symbolics 3675. Each file is listed in the Appendices at the end of this thesis. There are five files containing the major functions (classification utilities, slope utilities, contour utilities, B-spline smoothing utilities, and drawing utilities) and two files holding conversion functions and system constants. The five main function files require that the conversion functions file and the terrain constants file be loaded into the LISP world to execute properly. Additionally, the classification utilities file has several common functions which are utilized by the four remaining files. The easiest way to run the system is to login and load all seven of the files into the LISP world.

3. Getting Started

Before running any of the functions other than the drawing functions, it is wise to check on the setting of the constants. These constants can be found in the file **terrain-constants.lisp**. There are only four global constants. The curvature threshold is currently set at 0.01, and is the value for determining if there is significant curvature to classify a cell based on curvature. The dot product threshold is set at 0.01 and is used to see if the direction of maximum curvature and the direction of maximum slope are

orthogonal to each other. The slope limit is set at 2 degrees and is the value below which terrain is determined to be flat. The unsafe limit is set at 28 degrees and is the value above which the slope is determined to be unsafe. The only other value to check is the normalization matrix. This matrix is utilized to normalize the change in slope and curvature to the distance between data points. It is currently set for a distance of 12.5 meters (the distance between points in high resolution). If a file of low resolution grid is to be classified, this matrix would have to be changed to reflect the new distance between data points (100 meters). This matrix is not parameterized as it is felt that it would be unwise to classify low resolution data as most of the resolution would be lost for data over such a large distance. Over a distance of 100 meters, a number of terrain features which would be of significance could be skipped over or not detected.

4. Running the System

All of the major functions in the system are called with similar parametric input. Each requires three inputs. The first is **mapsize**. This is the number of elevation data points in either direction of the grid being analyzed. For a one kilometer high-resolution grid, this is 80. A two kilometer high-resolution grid would be 160. The second parameter is the **input file**. The files are currently stored on the Symbolics machine with a standard naming format. A typical elevation file would be named **fq5886.1-hr-e** for unsmoothed raw elevation data. The first part of the file name represents the standard UTM coordinate system representation for the lower left hand corner of a grid square. The information after the period indicates the file is one kilometer square, contains high resolution data and is for elevation data (as opposed to vegetation data). The final parameter is the **output file**. Output from the B-spline smoothing is typically **fq5886.1-hr-e-sm** to indicate that it is smoothed elevation data for the grid square being utilized. The typical output file from the classification, slope, and contour functions would look like **fq5886.class**, **fq5886.slope**, and **fq5886.cont**

respectively. Each of these output files is merely a list of codes corresponding to values determined by the main function. Rather than require the complex calculations to be performed each time a display of information is desired, the results of the calculations are converted to a code and this code is written to a file. This greatly facilitates later redisplaying of the information. This is typical of all the main functions in the system.

a. Smoothing the Terrain

The main function of this thesis is to first smooth the elevation data before running any of the other functions. After smoothing the raw elevation data, any of the functions can be run with either the smoothed or unsmoothed data as input to compare the results of the algorithms utilizing unsmoothed and smoothed data. To run the B-spline smoothing algorithm, the function must be called with the proper inputs. Since this is not a production system, there is no error handling and improper input will cause the programs to abort. This is standard with all the functions developed. A typical call of the B-spline smoothing function is (**run-smooth-elev 80 "fq5886.1-hr-e" "fq5886.1-hr-e-sm"**). As can be seen, the input to the function is a file of unsmoothed data and the output is a file of smoothed elevation data for the same grid square. At this point either file can be used as input to subsequent functions depending on the output desired. Each file has elevation data for all 6400 locations in the grid square.

b. Terrain Classification

After the raw elevation data has been smoothed with the B-spline smoothing algorithm, the next major step is the classification of the grid square with both the smoothed and unsmoothed data. The **run-class** function is the high level function which runs the classification algorithm. It is called just like the other high level functions with three parameters. A typical call of this function is (**run-class 80 "fq5886.1-hr-e-sm" "fq5886.class-sm"**). This function runs approximately three minutes for an individual grid square if the functions are compiled prior to execution. The result of

running this function is an output file which contains codes to represent all the possible types of terrain which have been classified. This facilitates display as the classifications do not have to be recalculated each time the display needs to be shown. The graphics function to display the classification is typically called each time the classification function is run, but it can be run separately after the grid square has been classified and reads the most recent output of the **run-class** function from a file and displays it on the color monitor. Three different 1km square test grids are so classified along with a 5km square grid.

c. Displaying Contours

The contour function has the same three parametric inputs as the classification function. A typical call of this function is (**run-contour 80 "fq5886.1-hr-e-sm" "fq5886.cont"**). This function makes no computation on the data but rather sends a code to the output file depending on the elevation data for each terrain cell. This contour information is then displayed on the monitor using color ramps to show the differences in elevation.

d. Slope Classification

The slope file also takes a file of elevation data as input and classifies each cell according to either edge, unsafe, flat, or safe. In the case of safe, it ramps the color depending on the degree of slope encountered. The parameters are similar to the other major functions and a typical call is (**run-slope 80 "fq5886.1-hr-e-sm" "fq5886.slope-sm"**). Again, this sends a code to the output file based on the appropriate slope of the individual terrain cell. The same basic information as the classification function is displayed with the difference being that the information concerning whether the cell is a ridge or ravine etc is eliminated. What is left is a display which shows the relative slope of each cell in the grid square regardless of the curvature. This type of information is useful in assigning weighted values to cells for route planning.

e. Drawing Functions

The drawing functions are called by each of the main functions (**run-class, run-slope**, and **run-contour**). However, they can be called separately to display any of the various forms of data previously calculated. There are four major functions in this section, **display-class, display-slope, display-cont,** and **display-veg**. Each of the functions has five similar parameters for input. The first is **mapsize** which has already been discussed. The second is the **input file** which is the name of the file containing the data calculated by one of the earlier functions. The third and fourth are **x-start** and **y-start** which are the starting point of the display in the x and y directions respectively. This allows the display to be moved to any desired location on the monitor. The final input is the **scale size**. This can be varied from one to any desired size. A scale of 12 overflows the screen for a standard 80 x 80 grid, so this becomes the logical maximum scale.

For a typical grid of 80 x 80, the system is designed to also display a scale to show what each of the colors depicted represent. This scale has to be displayed separately for sizes other than 80 and also when the scale size is less than 6 (when the character strings in the scale begin to run together). If an area larger than one grid square is being displayed, it is necessary to reduce the scale size in order to fit the entire display on the monitor. In such cases, it is necessary to display the scale for this type of display separately. Each of the scale functions for **classification**, **slope**, **contour**, and **vegetation** has a typical set of parameters as input.

The four scale display functions are **display-scale-class**, **display-scale-slope**, **display-scale-cont**, and **display-scale-veg**. There are five parameters required as input for each of these functions. The first two, **x-start** and **y-start**, are the beginning location of the scale display in the x and y direction respectively. This allows the scale to

be positioned at any desired location on the screen. The third parameter, **scale**, determines the height of the smaller boxes within the overall display. The fourth parameter, **b-width**, represents the width of the overall box being drawn as a background for the scale. The fifth parameter, **b-height**, is the overall height of the background box being drawn.

Finally, the vegetation display function is slightly different from the other three display functions. While the same number of parameters are required, the input for this file is the actual vegetation code which is extracted from the Hunter Liggett database. There are no calculations to be made on this data since it is already in the form of a code between zero and seven, so the input file name is in the same format in which it was extracted from the digital database. The format for a typical input file for this function is **fq5886.1-hr-v**.

H. SUMMARY

In this thesis, a number of algorithms are developed to enhance the ability to classify and display information about a grid square. Each group of related functions is in a separate file for ease of access and modification. The running of the functions is relatively simple once a user has become accustomed to the peculiarities of LISP and the Symbolics machine. The main additions to the previous work are the B-spline smoothing functions, the slope functions, and the contour functions. The graphics drawing functions have also been expanded to provide increased flexibility. The next chapter presents typical results obtained with this terrain analysis program.

V. EXPERIMENTAL RESULTS

A. INTRODUCTION

The research completed before this thesis has been expanded from a single display of classification data to a system which is capable of smoothing raw elevation data and displaying four types of information for analysis and comparison. The four different displays which can be shown on the color monitor depict a large amount of valuable data already extracted from a simple digital database. With a small amount of calculation, a variety of information has been developed on numerous terrain cells. This information can eventually be further refined, expanded, and coordinated to make it even more meaningful to future users or autonomous vehicles.

B. TYPICAL DISPLAYS

The system which is currently running on the Symbolics machine has four outputs available for display. These four displays are the **Classification** display, **Slope** display, **Contour** display, and **Vegetation** display. Each can be shown anywhere on the screen with any scale size from one to a practical upper limit of 11 which will fill the typical screen. The outputs can be displayed individually or in any combination to better enhance understanding and comparison of the available information.

While there are four separate displays which are output by the system, by far the most meaningful at this point is the classification display. This display has a wealth of information and is the most computationally intensive of the four functions. Each of the four displays shows the data in a format that is similar, representing calculated data for each of the 6400 cells in a standard grid square.

1. Classification Display

The information in this display is the most relevant to the attempts to classify terrain into regions. It shows not only the classification, but also the degree of slope in planar sections of the display. This type of information is extremely useful for route planning since a different value can be associated with each of the various types of terrain identified. The information is color coded for ease of display and understanding. The 18 different classifications are detailed in Table 5.1. The upper portion of Figure 5.1 is a typical display of a one kilometer grid square showing the various types of classification which are possible. This display is unique and to the trained observer begins to mirror the analysis of elevation data that a human would reach through visual scanning of actual terrain and a topographical map. However, unlike a human, the machine is capable of storing and recalling this information over a wider area with a much higher degree of accuracy.

In the upper middle portion of the display a small hill is depicted. It is capped by a **peak** and has steeply sloping sides. At the bottom of the hill, just before the terrain flattens out, **ravine** cells appear which almost encircle the entire hill. This is typical as the sloping terrain changes from steep to flat. In the lower left of the display another hill appears, but this time the sides of the hill are so steep as to make them **unsafe**, thus the appearance of the red cells. However, the top of the hill is again depicted by the appearance of **peaks** with several **ridges** leading down from the top. Throughout the grid square the appearance of terrain features are detected which correspond to the features encountered on a standard topographical map. In this case, however, the information is in a format which is more relevant to route planning and is more likely to be understood by a machine than the typical contour lines.

Table 5.1

TERRAIN CLASSIFICATION TYPES

Classification	Color
R-l Ridge	Purple
R-l Ravine	Blue-green
Peak	Black
Pit	Dk Blue
Ridge	Gray
Ravine	Blue
Saddle	Lt Gray
Pass	Lt Blue
Flat	Green
Planar 25-28 degrees	ExDk Yellow
Planar 21-25 degrees	VDk Yellow
Planar 17-21 degrees	Dk Yellow
Planar 13-17 degrees	Yellow
Planar 9-13 degrees	Lt Yellow
Planar 5-9 degrees	VLt Yellow
Planar 2-5 degrees	ExLt Yellow
Edge	Maroon
Unsafe	Red

2. Slope Display

The lower half of Figure 5.1 shows a typical display of slope classification for the same grid square. The classification types for the slope display are depicted in Table 5.2. There are only ten possible classifications (flat, unsafe, edge, or one of seven possible safe slopes greater than 2 degrees but less than 28 degrees). As can be seen, the slope and classification displays are quite similar. The major difference is that the slope display is only concerned with the maximum slope of each terrain cell and makes no distinction based on the curvature. Thus a cell that is displayed as a **ridge** (gray) on the

Table 5.2

SLOPE CLASSIFICATION TYPES

Classification	Color
Flat < 2 degrees	Green
Safe 2-5 degrees	ExLt Yellow
Safe 5-9 degrees	VLt Yellow
Safe 9-11 degrees	Lt Yellow
Safe 13-17 degrees	Yellow
Safe 17-21 degrees	Dk Yellow
Safe 21-25 degrees	VDk Yellow
Safe 25-28 degrees	ExDk Yellow
Unsafe > 28 degrees	Red
Edge	Maroon

classification display, appears on the slope display as one of seven values based on its maximum slope. The small hill in the upper middle of the grid square is now depicted only as a range of slope values depending on the steepness of each cell.

It is significant to note in comparing the two displays that all of the terrain cells that are **flat** on the slope display are likewise **flat** on the classification display. The **flat** cells on the slope display all have slopes less than 2 degrees. These same cells on the classification display can be displayed as one of the other classification types if they have sufficient curvature along either the major or minor axes. Since none of the cells that are below 2 degrees on the slope display have a secondary classification on the classification display, it can be concluded that when a cell has less than 2 degrees of slope along its major axis, there is insufficient curvature to allow classification based on a **secondary** class. While this is not a rigorous mathematical proof, there is sufficient evidence to allow for generalization. Unique conditions might occur which would allow a single cell

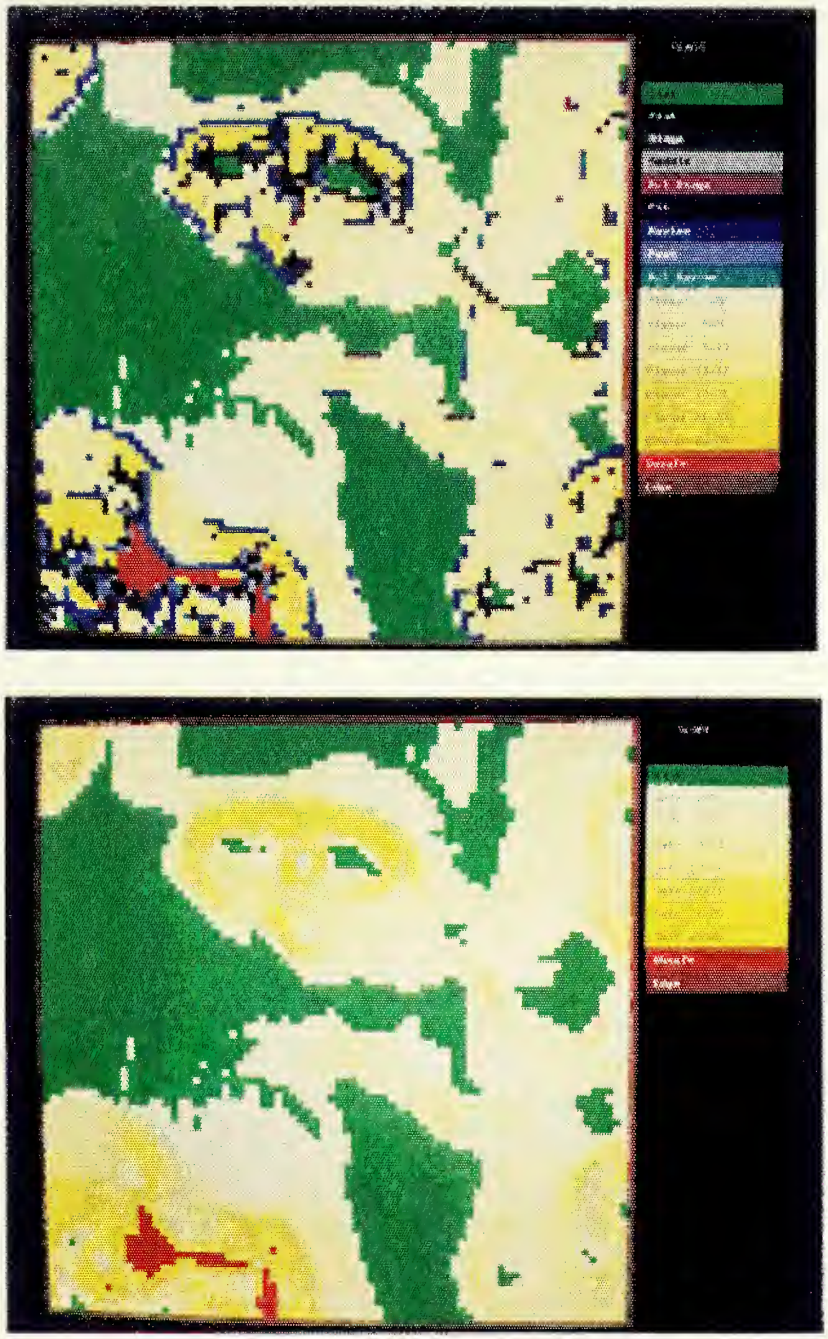

Figure 5.1 Typical Terrain Classification and Slope Display - fq5580

to have a slope less than 2 degrees and still have a **secondary** classification. However, this does not materially alter the overall effect. Consequently, any future work in this area could increase the computational efficiency of the algorithm by eliminating the attempt to classify a cell based on curvature if the **primary** classification is **flat**.

The shading of the colors in the safe slope range is not easily distinguished by the human eye. However, this information is readily distinguished by a computer as it utilizes the values from a bit map for the display to quickly determine the exact color being represented. This type of information is most useful as input to a route planning routine as each cell can easily be given a weight based on the available information to assist in searching for the best possible route. The display itself mirrors the classification display and is a merely a second method of storing and displaying data about the grid square involved without making a number of additional calculations.

3. Contour Display

The contour display is most useful as a tool for checking to see that the classification information is correct. It is a quick method of displaying contours to depict changes in contour intervals. With this type of information, it is possible to estimate where ridges, ravines, peaks, pits, and other topographical features occur and compare the estimated results with the output from the classification algorithm. Table 5.3 shows the various classifications associated with the contour display. The information here is based entirely on the elevation of each cell. The upper half of Figure 5.2 depicts a typical contour display for a selected grid square. This display is also useful in comparing the elevations from a topographical map to the system elevation data to ensure that there are no major errors or omissions in the raw data. As was discussed earlier, the data in the database is highly accurate, but there are possibilities that errors in extraction can occur or that the information in the database is erroneous. By utilizing

Table 5.3

CONTOUR CLASSIFICATION TYPES

Classification	Color
<1000	Red
1000-1040	VDk Blue
1040-1080	Dk Blue
1080-1120	Blue
1120-1160	Lt Blue
1160-1200	VLt Blue
1200-1240	VDk Green
1240-1280	Dk Green
1280-1320	Green
1320-1360	Lt Green
1360-1400	VLt Green
1400-1440	VDk Red
1440-1480	Dk Red
1480-1520	Red
1520-1560	Lt Red
1560-1600	VLt Red
1600-1640	VDk Brown
1640-1680	Dk Brown
1680-1720	Brown
1720-1760	Lt Brown
1760-1800	VLt Brown
>1800	Orange

this display a user can verify that for the selected grid square, the digital data matches what was expected based on the topographic map. It is also possible to achieve a feeling for anticipated ridges, ravines, peaks, and pits.

In Figure 5.2, the small hills in the middle and lower left of the grid square are easily recognized. Additionally, the rapid change in contour colors in the lower left of the screen depict a steeply sloping region. When this is compared to the classification

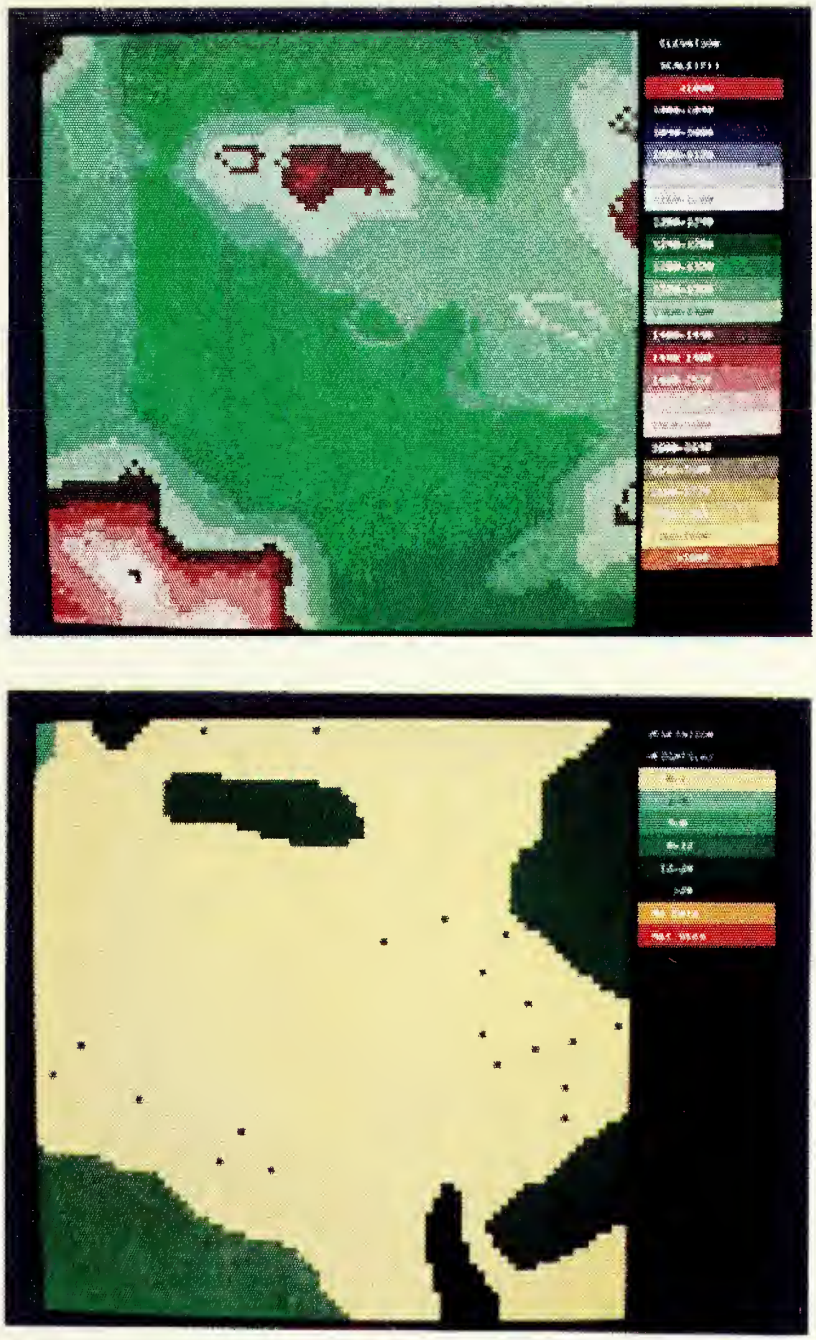

Figure 5.2 Typical Contour and Vegetation Displays - fq5580

and slope displays in Figure 5.1, it is readily apparent that the data is accurate and depicts the proper values for each area of the grid square. During development of the algorithms, constant use was made of this display in conjunction with the topographical map in checking to ensure that the classification and slope algorithms were functioning properly. From this analysis it is apparent that the system is functioning properly and that the sections of terrain are being properly analyzed.

A final analysis of the contour display reveals some of the minor inconsistencies previously mentioned which this thesis attempts to eliminate. On the hill in the upper portion of the display, there are two distinct peaks. This is confirmed by the topographic map. However, the red circle on the left of the hill looks like a basin since the area inside the circle changes to green (the lower elevation). In fact, the area in the circle should be red. This is an example of small pieces of data that are probably in error and which potentially can be adjusted by the B-spline smoothing algorithm.

4. Vegetation Display

The typical vegetation display in the bottom half of Figure 5.2 is based on one of eight codes from the database which represents varying heights of vegetation on a particular terrain cell. Table 5.4 shows the meaning of each possible code. The information depicted is relatively accurate and is comparable to the various color schemes utilized on the standard topographical map to represent vegetation. It is apparent, however, that in arriving at the codes, a method more sophisticated than extracting from the contour map was utilized. The contour map does not contain sufficient detail to discern a variance of several meters as the codes in the database suggest. The best possible explanation is that aerial photographs were utilized in conjunction with the maps to arrive at the desired codes. This is significant in that this is another possible method of improving classification techniques. By utilizing aerial

Table 5.4

VEGETATION CLASSIFICATION TYPES

Classification	Color
0-1 meter	Tan
1-4 meters	VLt Green
4-8 meters	Lt Green
8-12 meters	Green
12-20 meters	Dk Green
>20 meters	VDk Green
No Data Available	Orange
Not Used	Red

photos in conjunction with existing maps it is possible to refine and update the information available and more accurately depict the data as it actually exists. This, however, is left for future work, and is not in the scope of this research.

At this point, this information does not have a large amount of significance in the overall project of classifying individual terrain cells. However, future work could make excellent use of this information. One of the uses of this system, as already discussed, could be in route planning. One of the major considerations is obviously the classification of a particular cell. If an expert system were working with the various types of information available, it might be able to incorporate not only the topographic classification data, but also the vegetation data and likewise any tactical information that was available. So, while this display is only of moderate use now, it could have significant impact in future systems.

C. COMPARISON OF DISPLAYS

The primary concern of this thesis is the use of B-splines to smooth the raw elevation data and determine just what effect this has on the ability of the algorithms to properly classify terrain. Other than the modifications already mentioned, no major work has gone into adjusting the concept utilized in classifying the terrain cells. A quadratic fit is applied to each elevation point and the eight surrounding points. After taking the first and second derivatives of this mathematical representation, the slope and curvature are utilized to determine the type of cell encountered. Before this work is begun, however, it is necessary to first smooth the elevation data.

1. Smoothing of Elevation Data

The algorithms to smooth the raw elevation data work as expected. From Figure 5.3 the contour intervals before and after application of the B-spline smoothing algorithms can be seen. The upper portion is a contour display of the raw elevation data. The lower portion depicts the contour display for the same grid square after the B-spline smoothing algorithm has been applied. While no major effects on the contours are evident, by looking at selected points it can be noted that minor inconsistencies are eliminated. Several cells previously encircled by different color cells are smoothed to the same approximate elevation. The minor problem on the finger in the upper center of the grid has also been corrected. These types of adjustments have definite advantages later on, but it should also be noted that the B-spline algorithms (and any of the other surface representation algorithms) tend to smooth out peaks and pits and also to reduce some of the finer definitions in the surface. This is useful when dealing with a minor flaw in the data, but has some undesired effects in eliminating accurate terrain depictions.

2. Classification

The upper portion of Figure 5.4 shows the classification of grid square **fq5886** before any smoothing had been done on the elevation data. The lower portion shows the

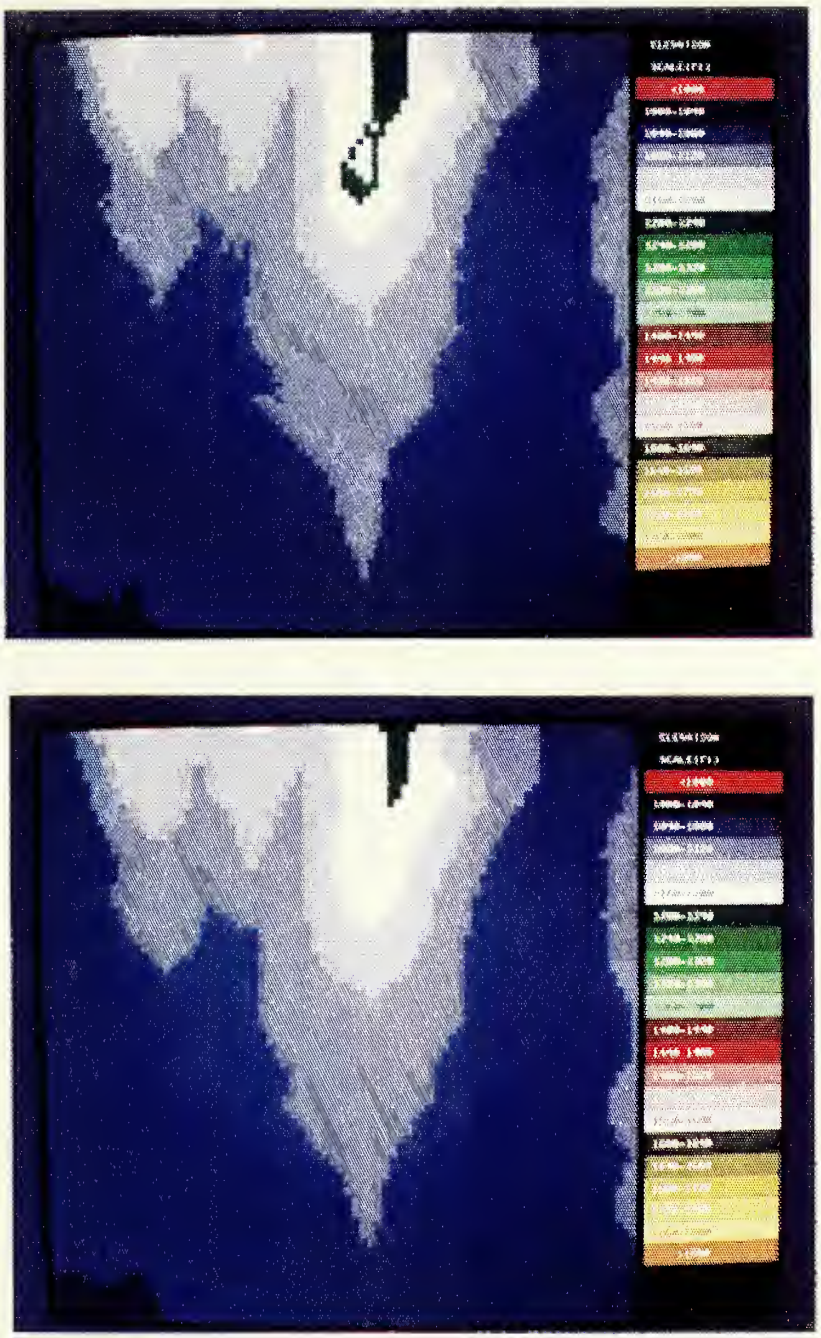

Figure 5.3 Unsmoothed and Smoothed Contour Displays - fq5886

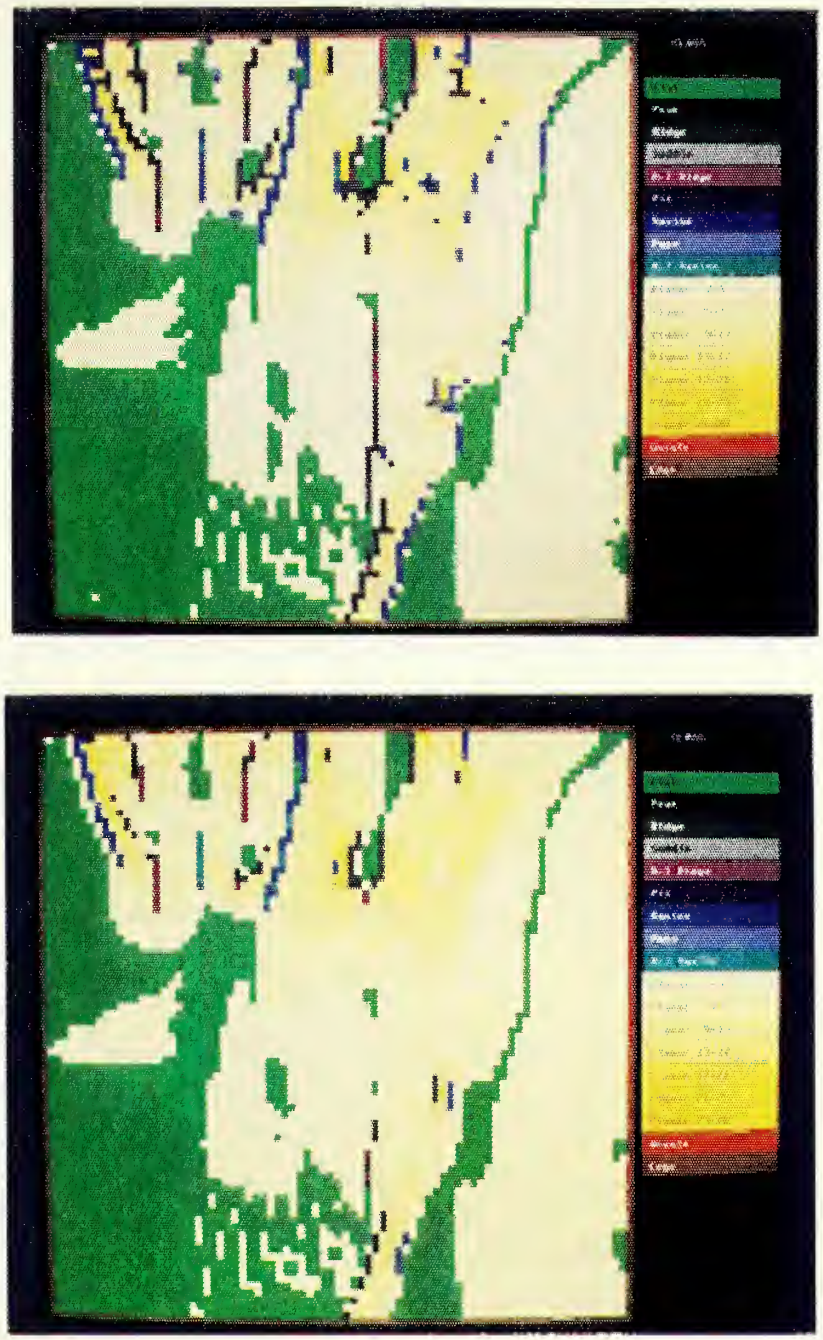

Figure 5.4 Unsmoothed and Smoothed Classification Displays - fq5886

same grid square after the B-spline smoothing algorithm has been applied to the raw data. It can be seen that the upper display is a fairly accurate representation of the grid square in question. By looking at the topographical map or the elevation display in Figure 5.3, the trained individual can see that the classification algorithm has detected what should be several ridges and ravines running basically north and south in the grid square. This is what was expected and is fairly accurate. What was not expected was that there would be gaps in the ridge and ravine lines. A proper algorithm was expected to depict ridge lines and ravine lines connected continuously and also to basically meet somewhere on the display thus dividing the entire region into a number of *watersheds.* The fact that it failed to do this is not totally disappointing but this is where it was believed that the use of the B-spline would smooth out some of the raw data and provide a better depiction of the grid square. Thus, the lower half of the figure shows the classification after the data had been smoothed and the same algorithms applied to the smoothed elevation data.

The results are certainly not as dramatic as expected. The new classification display shows the same-north south orientation of the actual ridges and ravines in the grid square, but this definition is not as complete as in the original display. Some of the excess classifications have been eliminated, but so have some of the desired classifications. This is most likely a result of the B-spline smoothing algorithm having a tendency to level out ridges and ravines and reduce some of the critical definition in a cell. It is also likely, however, that these little missing pieces are not capable of being classified with the existing algorithms and need some sort of expert system wrapped around the entire process to help determine actual classifications.

A final possibility is that the chatter encountered in the display is valid. When dealing with ideal test elevation data it is possible to achieve very good results.

However, an actual terrain database is bound to have a number of flaws in the data in addition to small pieces of terrain which are inadvertently placed around the grid square, and have sufficient curvature to allow the algorithm to classify them. Certainly, one possibility would be to classify groups of cells and determine if a cell is in fact a part of a larger definition before actually classifying it.

3. Threshold Adjustments

Since the overall effect of smoothing the data had been to reduce the number of cells which had had sufficient curvature to allow the algorithm to classify them, an attempt was made to reduce the curve threshold to allow more cells to be classified. Figure 5.5 shows the effects of reducing the curve threshold from 0.01 to 0.004. A marked increase in the number of cells which can be classified is apparent. The ridges and ravines which run basically north-south in this grid square are again more accurately depicted. The difference in the two displays is that the upper display has a dot-product threshold of 0.01 while the lower display has a dot-product threshold of 0.001. The differences are easily visible. It can be seen that allowing a greater dot-product threshold increases the number of cells classified with the one method. There is no effort here to determine which is the more accurate method of terrain classification. This type of information might be useful to some later research.

Figure 5.6 shows the same basic information. However, the curve threshold has now been set to 0.006. The dot-product threshold in the upper display is 0.01 while the lower display has a dot-product threshold of 0.001. What is noticeable here is that the number of individual cells which can now be classified is dramatically increased. A definite widening of the ridges and ravines is visible. Of course this is not altogether good. The intent is not to so reduce the threshold as to allow for each cell to be classified as one of the six basic types (Peak, Pit, Ridge, Ravine, Saddle, and Pass). The intent is to find large areas of terrain with common characteristics which are separated by ridges and

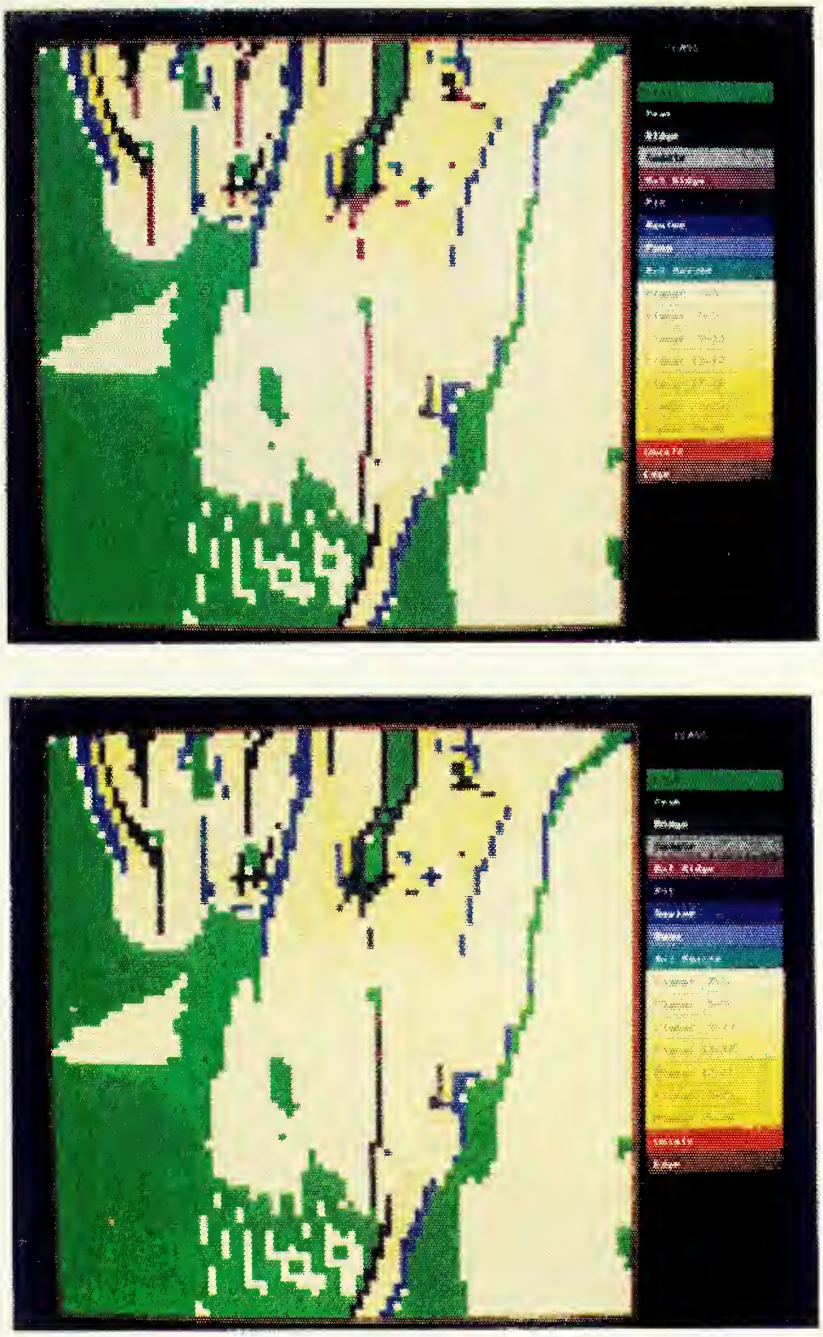

Figure 5.5 Smoothed Classification Display - Curve 0.004 - fq5886

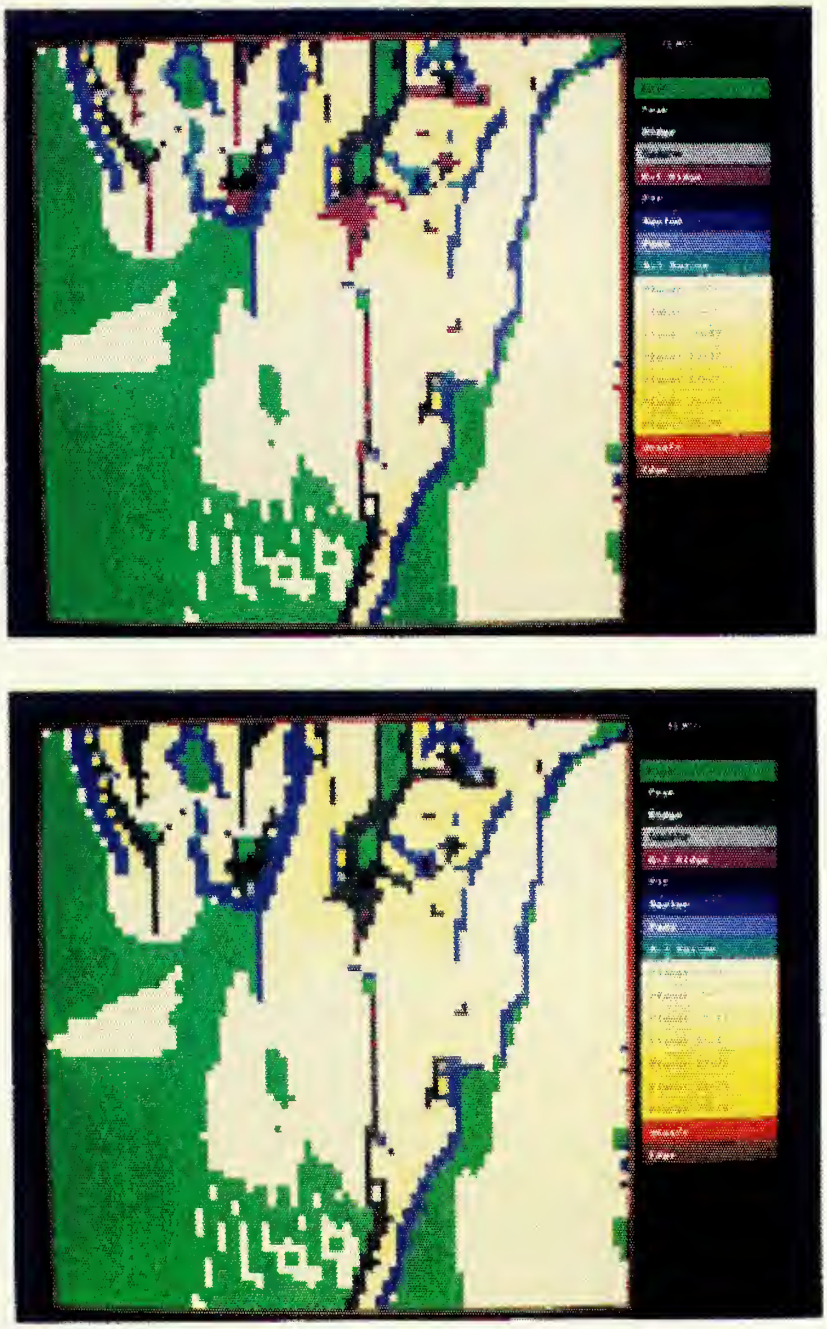

Figure 5.6 Smoothed Classification Display - Curve 0.006 - fq5886

ravines. By so doing, the terrain can be segmented into areas with a constant traversal cost. This would avoid the requirement for calculating individual cost factors for each cell while a search algorithm was traversing the terrain.

Another consideration in determining the proper thresholds is the overall spectrum that actual terrain can span. The grid squares already analyzed and discussed were relatively flat with a minimum of terrain features which made classification easy. As the thresholds were lowered, recognition was increased and better definition of ridges and ravines was accomplished. What happens, however, when a piece of terrain is steeply sloped and already has a high degree of definition?

Figure 5.7 shows a classification display of a five kilometer square area. In the 25 grid squares in this display the terrain varies from almost entirely flat to steeply sloped. This area was classified with a curve threshold of 0.01 and a dot-product threshold of 0.01. Lowering either threshold would increase classification. However, in some areas this would be undesirable. In the steeply sloped grids in the upper center of the display there is already sufficient classification to recognize ridges, ravines and other features. In fact there is already more classification than is essential. Sections of terrain contain an excess of cells which have been classified when in fact they are contained in a terrain segment where all the cells are of approximately the same type. In this region, an expert system might be utilized not to close the segmentation areas, but rather to eliminate unnecessary classifications in segmented regions. On the other hand, there are several flat areas in the lower center of the grid square, where the thresholds would have to be significantly reduced to achieve any sort of classification. In this region, an expert system might be utilized to connect some of the segments where closure has not been achieved. A lowered threshold for either curvature, dot-product, or both would tend to provide too much information at times and not enough at other times.

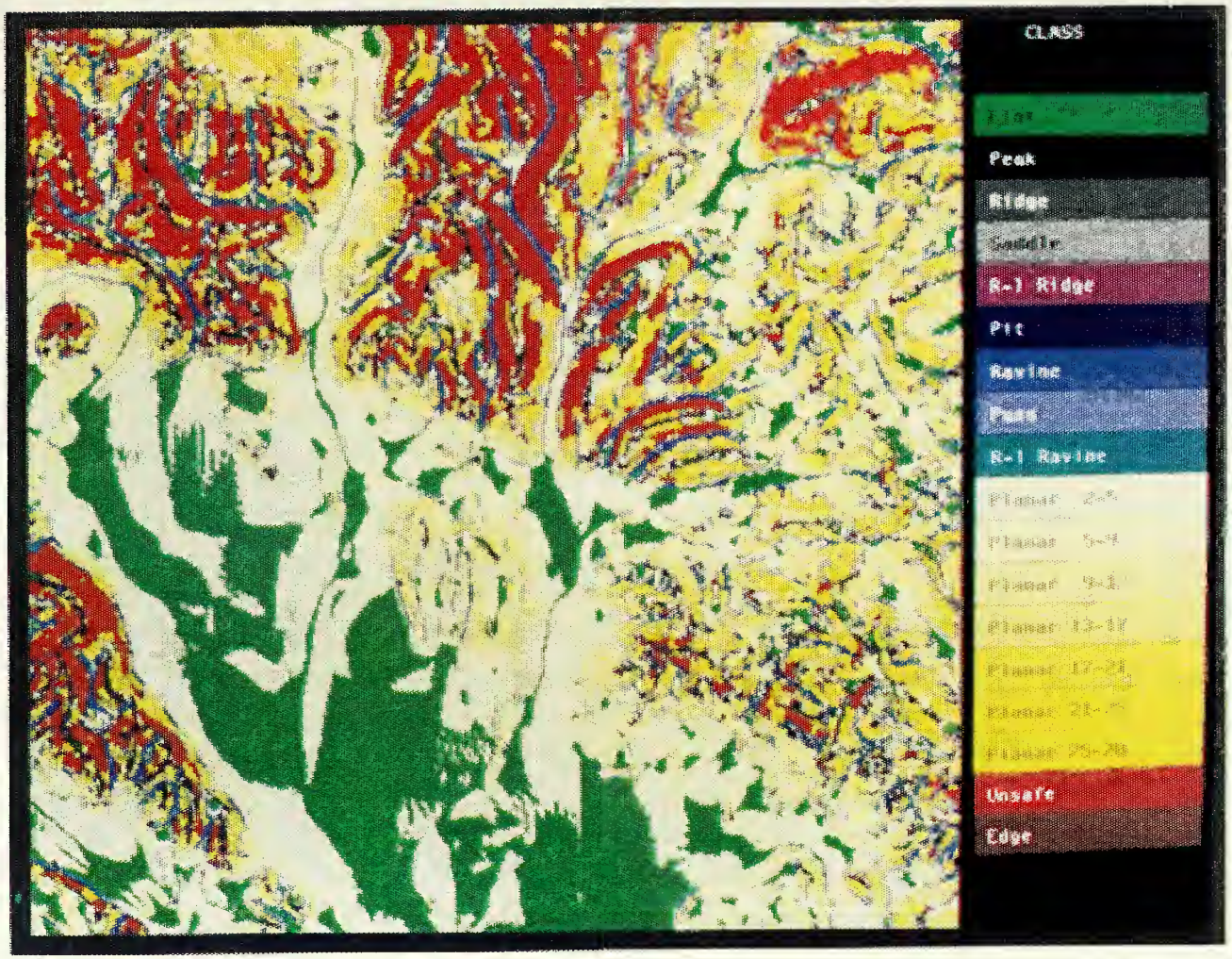

Figure 5.7 Five kilometer square Classification Display - fq5685

After working with a number of grid squares, it appears that the settings used for the original algorithm are sufficient. These limits allow for classification of terrain across a wide variety of terrain types without over-classifying any region. Changing the thresholds might help in relatively flat areas but it would be a detriment to proper recognition in other more steeply sloped grid squares.

D. SUMMARY

While it was apparent that the algorithms work as expected, the results are less than spectacular. The same smoothing algorithms that are intended to eliminate small errors and variances also have a tendency to flatten the surface and eliminate some key features necessary for accurate classification. What has been achieved is the inclusion of the B-spline smoothing algorithm, which is readily adapted to work in this environment. The type of information available has been expanded and demonstrates the wide variety of information which can be developed from the dense elevation data in the database. Further work will reveal just how helpful this has all been.

VI. SUMMARY AND CONCLUSIONS

A. RESEARCH CONTRIBUTIONS

This thesis is a continuation of work which is all part of a major research effort to enhance the capabilities of autonomous vehicles, although the concepts developed here can also be useful to manned vehicles. The overall mobility research activity at the Naval Postgraduate school deals with a myriad of functions for autonomous vehicles and is centered around the development of a six legged walking machine, including leg motion and coordination, close-in terrain scanning, and route planning, to name a few of the many areas of work [Ref. 15]. While the techniques developed and used here are only a small part of the overall picture, they are just as critical as any part of the project since the whole machine is only as good as its weakest link.

This thesis employs a graphics tool, B-splines, to enhance the capabilities of a computer to analyze and recognize terrain features in an effort to improve route planning and terrain recognition. It demonstrates that the use of spline curves can be of significance in working with elevation data. Splines can be utilized to smooth the data and eliminate possible errors in the data. However, because of the tendency of splines to eliminate peaks and valleys and smooth specific features, they do not materially facilitate the classification of individual cells.

B. RESEARCH EXTENSIONS

Future work in this area could be centered around efforts to either further refine the algorithms to enhance the ability to classify individual cells or to provide additional expert system support for improving the information already extracted. Additional research could take several approaches to improving the algorithms.

First, the minor inconsistencies that occur in the elevation data and classifications could be nothing more than the variances in nature itself. If so, then these types of variances should remain. There is no guarantee that once a ridge is identified it will continue smoothly down a finger of land until it connects with a ravine or saddle. A skilled individual walking the ground might be able to detect the general pattern of the ridge, but there certainly could be places where the terrain flattens out below an arbitrarily selected curve threshold which would make accurate classification almost impossible. This is a shortcoming of any purely computational system. There must be mathematical limits that cause some type of error.

The greatest promise then seems to be some sort of expert system that employs not only the complex mathematical calculations in classifying terrain, but also includes a high degree of knowledge about the tendencies of terrain. If a ridge has been running downhill for 50 meters and then suddenly stops on one side of a grid cell, only to continue on the other, it might be valid to assume the ridge ran through the blank cell but the curvature was below a given threshold which would have allowed for proper classification. Such an expert system would not be perfect, but could go a long way towards eliminating many of the minor inconsistencies found in the classification displays.

An effort might be made to utilize the B-spline surface approximation in the classification algorithm rather than utilizing it to first smooth the terrain and then using quadratic equations to classify the terrain. It is possible that, for the purpose intended, the raw elevation data is as good or better than could be expected. A number of possibilities exist if this is the alternative explored. One alternative could be the use of higher order splines, although this possibility might force the algorithms to become too computationally intensive.

No matter what the alternative, there is is a wide range of possibilities to explore as a follow on to this work. The use of B-splines has demonstrated the usefulness of existing graphics functions to the terrain data. While the results have not been spectacular, the next effort in this area could find a more complex method which would significantly enhance terrain recognition. Ultimately, as the speeds of processors improve and the classification algorithms improve, it will be possible for a machine to begin to make better use of the millions of bits of data at its disposal much quicker than a human being. The long range purpose of teaching a machine to cross terrain autonomously under a number of both technical and tactical constraints will no doubt eventually be achieved.

LIST OF REFERENCES

1. Benoit, E., "Plotting Complexity," *Forbes*, v. 136, no. 7, pp. 182-3, Sept 16, 1985.
2. Goodpasture, B. K., *Terrain Classification From Digital Elevation Data Using Slope and Curvature Information,* M. S. Thesis, Naval Postgraduate School, Monterey, California, December 1987.
3. Haralick, R. M., Watson, L. T., and Laffey, T. J., "The Topographic Primal Sketch," *The International Journal of Robotics Research*, v. 2, no. 1, pp. 50-72, Spring 1983.
4. Douglas, D. H., "Experiments to Locate Ridges and Channels to Create a New Type of Digital Elevation Model," *Cartographica*, v. 23, no. 4, pp. 29-60, Winter 1986.
5. Poulos, D. D., *Range Image Processing for Local Navigation of an Autonomous Land Vehicle,* M. S. Thesis, Naval Postgraduate School, Monterey, California, September 1986.
6. Fu, K. S., Gonzalez, R. C., and Lee, C. S. G., *ROBOTICS: Control, Sensing, Vision, and Intelligence,* pp. 331-360, McGraw-Hill Book Company, New York, New York, 1987.
7. Yee, S. H., *Planar Patch Terrain Modeling,* M. S. Thesis, Naval Postgraduate School, Monterey, California, June 1988.
8. Richbourg, R. F., *Solving a Class of Spatial Reasoning Problems: Minimal-Cost Path Planning in the Cartesian Plane,* Ph.D. Dissertation, Naval Postgraduate School, Monterey, California, June 1987.
9. Hearn, D. and Baker, M. P., *Computer Graphics,* Prentice-Hall, Englewood, New Jersey, 1986.
10. Pavlidis, T., *Algorithms for Graphics and Image Processing,* Computer Science Press, Rockville, Maryland, 1982.
11. Foley, J. D. and Dam, A. Van, *Fundamentals of Interactive Computer Graphics,* Addison-Wesley Publishing Company, Reading, Massachusetts, 1982.
12. Bartels, R. H., Beatty, J. C., and Barsky, B. A., *Splines for Use In Computer Graphics and Geometric Modeling,* pp. 46-56, Morgan Kaufmann Publishers, Inc., Los Altos, California, 1987.
13. Anon., *IRIS User's Guide,* v. I, p. 11-28, Silicon Graphics, Inc., Mountain View, California, 1987.
14. Zyda, M. J., McGhee, R. B., and Taylor, G. W., *Parametric Representation and Polygonal Decomposition of Curved Surfaces*, U.S. Naval Postgraduate School Report NPS52-86-028, pp. 20-22, December 1986.
15. Kwak, S. H. and McGhee, R. B., *Rule-Based Motion Coordination for the Adaptive Suspension Vehicle*, U.S. Naval Postgraduate School Report NPS52-88-011, May 1988.

APPENDIX A - CONVERSION FUNCTIONS

```
;;; -*- Mode: LISP; Syntax: Common-lisp; Base: 10 -*-

;*****************************************************************
;*                    conversion-factors.lisp
;*****************************************************************
;*****************************************************************
;*  This file contains several conversion functions.
;*
;*  These functions are called by (run-class mapsize datafile classfile)
;*  (run-slope mapsize datafile slopefile) and (run-smooth-elev mapsize
;*  datafile elevfile).  This file must be included in the Lisp world
;*  in order for any of these functions to execute.
;*****************************************************************
;*****************************************************************

(defun feet-to-meter (feet)
  (* feet 0.3048))


(defun meter-to-feet (meter)
  (* meter 3.28084))


(defun rad-to-degree (radians)
  (/ radians 0.0174533))


(defun degree-to-rad (degrees)
  (* degrees 0.0174533))


(defun arctan (y x)
  (cond
    ((or (and (minusp y) (minusp x))
         (and (plusp y) (plusp x)))
     (rad-to-degree (atan (abs y) (abs x))))
    (t (- (rad-to-degree (atan (abs y) (abs x)))))))
```

APPENDIX B - TERRAIN CONSTANTS

```
;;; -*- Mode: LISP; Syntax: Common-lisp; Base: 10 -*-

;*****************************************************************
;*                terrain-constants.lisp
;*****************************************************************
;*****************************************************************
;* This file contains the following constants
;*
;* 1. curv-threshold - The amount of leadway that the eigenvalue has
;*    from zero when classifying the secondary classification.
;* 2. dot-prod-threshold - The threshold for determining two items
;*    are orthogonal.
;* 3. slope-limit - The upper limit of the slope when classifying level
;*    terrain.
;* 4. unsafe - The upper limit of the slope when classifying safe
;*    terrain, any slope above this value is considered unsafe.
;* 5. A-matrix-u, AT-matrix-u, ATA-matrix-u, ATAI-matrix-u, B-matrix-u -
;*    Various matrices which are the steps necessary to calculate
;*    the B-matrix-u which allows us to calculate the K-matrix.
;* 6. norm-matrix - Matrix used to normalize the slope and curvature.
;* 7. b-spline-matrix - This is the basis matrix required for
;*    calculating the smoothed elevations.
;* 8. b-spline-matrix-t - The transpose of the b-spline basis matrix.
;*****************************************************************
;*****************************************************************

(setq curv-threshold 0.01)

(setq dot-prod-threshold 0.01)

(setq slope-limit 2.0)

(setq unsafe 28.0)

(setq A-matrix-u (make-array '(9 6) :initial-contents
                    '((1 -1  1  1 -1  1)
                      (1  0  1  0  0  1)
                      (1  1  1  1  1  1)
                      (1 -1  0  1  0  0)
                      (1  0  0  0  0  0)
                      (1  1  0  1  0  0)
                      (1 -1 -1  1  1  1)
```

```
                    (1  0 -1  0  0  1)
                    (1  1 -1  1 -1  1))))


(setq AT-matrix-u (math:transpose-matrix A-matrix-u))
(setq ATA-matrix-u (math:multiply-matrices AT-matrix-u A-matrix-u))
(setq ATAI-matrix-u (math:invert-matrix ATA-matrix-u))
(setq B-matrix-u (math:multiply-matrices ATAI-matrix-u AT-matrix-u))

(setq norm-matrix (make-array '(6 6) :initial-contents
                    '((0.08 0 0 0 0 0)
                     (0 0.08 0 0 0 0)
                     (0 0 0.08 0 0 0)
                     (0 0 0 0.0064 0 0)
                     (0 0 0 0 0.0064 0)
                     (0 0 0 0 0 0.0064))))

(setq b-spline-matrix (make-array '(4 4) :initial-contents
                      '((-1/6  3/6 -3/6  1/6)
                       ( 3/6 -6/6  3/6  0  )
                       (-3/6  0    3/6  0  )
                       ( 1/6  4/6  1/6  0  ))))

(setq b-spline-matrix-t (math:transpose-matrix b-spline-matrix))
```

APPENDIX C - TERRAIN CLASSIFICATION FUNCTIONS

```
;;; -*- Mode: LISP; Syntax: Common-lisp; Base: 10 -*-

;*****************************************************************
;*                    class-utilities.lisp
;*****************************************************************
;*****************************************************************
;* List of Functions in this file
;*
;* 1. run-class - Top function to run the digital terrain classification.
;*    Inputs are mapsize which is the size of the array from the datafile,
;*    i.e. for high resolution the array would be 80 x 80, so mapsize
;*    would be 80. The array must be a square to work in this program.
;*    The datafile is the input file and it must be in double quotes.
;*    The classfile is the output file and it also must be in double
;*    quotes. Sample call (run-class 80 "fq5886.1-hr-e" "fq5886.class").
;* 2. make-tc-map - Makes an array with the name hunter and of the size
;*    given by the user.
;* 3. load-tc-attribute - Loads the array from the datafile with the slot
;*    given. In the case of this program, it loads elevation, but other
;*    slots could be loaded.
;* 4. load-tc-map - Loads the array with all necessary mathematical data
;*    and also primary and secondary classifications according to the
;*    eigenvalues of the Hessian matrix. In this example, the array is
;*    called hunter but it could be changed.
;* 5. calculate-eigenvalues-and-slope - This function is used by the
;*    load-tc-map function. Its inputs are five of the constants of
;*    the quadratic equation and it outputs a property list that is
;*    appended to each spot in the array.
;* 6. classify - This function is called by load-tc-map and it
;*    classifies the pixel according to the eigenvalues and the slope.
;*    This information is the appended to the property list in each
;*    spot in the array.
;* 7. calculate-gradient - This function is called by load-tc-map and
;*    it calculates the gradient which is appended to the property
;*    list in each spot in the array.
;* 8. rline-test - Uses data from the array by accessing its property
;*    list. From this it determines if the pixels is a ridge or ravine
;*    line. This information is then placed in the property list.
;* 9. write-classification - Takes the array and outputs the desired
;*    information to the classfile which is designated by the user.
;*    The output is in bitmap form and may be changed to represent any
;*    desired data that the user wants.
```

```
;* 10. get-tc - This function allows the user to see the property list
;*     associated with a particular spot in the array. The user must input
;*     the terrain-cell-map, in this case, hunter and the x-coordinate
;*     and the y-coordinate of the desired spot.
;* 11. get-tc-group - This function allows the user to see a specific list
;*     associated with a given property of a 3 x 3 square of the array.
;*     The terrain-cell-map is again named hunter and the x-coordinate/
;*     y-coordinate are of the middle cell of the 3 x 3 square.
;* 12. edge-tc-p - Tests for edge pixels
;* 13. corner-tc-p - Tests for corner pixels
;* 14. range-tc-p - Tests to see if the y-coordinate and x-coordinate are
;*     within the size of the array.
;*****************************************************************
;*****************************************************************


(defun run-class (mapsize datafile classfile)
 (make-tc-map hunter mapsize)
 (load-tc-attribute hunter datafile 'elevation)
 (load-tc-map hunter)
 (rline-test hunter)
 (write-classification hunter classfile)
 (send *blue-window* :refresh)
 (display-class mapsize classfile 70 50 11))


(defmacro make-tc-map (terrain-cell-map mapsize)
 (list 'setq terrain-cell-map (list 'make-array (list 'list mapsize mapsize))))


(defun load-tc-attribute (terrain-cell-map datafile slot)
 (setq input-stream (open datafile :direction :input))
 (let ((mapsize (sqrt (array-total-size terrain-cell-map))))
 (do ((xcoord 0 (1+ xcoord)))
    ((= xcoord mapsize))
   (do ((ycoord (1- mapsize) (1- ycoord)))
      ((< ycoord 0))
    (setf (aref terrain-cell-map xcoord ycoord)
        (cons slot (cons (read input-stream) (aref terrain-cell-map xcoord ycoord)))))))
 (close input-stream))


(defun load-tc-map (terrain-cell-map)
 (let ((mapsize (sqrt (array-total-size terrain-cell-map))))
  (do ((xcoord 0 (1+ xcoord)))
```

```
    ((= xcoord mapsize))
  (do ((ycoord 0 (1+ ycoord)))
    ((= ycoord mapsize))
   (cond
    ((edge-tc-p mapsize xcoord ycoord)
     (setf (aref terrain-cell-map xcoord ycoord)
         (append '(gradient1 egde
                   gradient2 edge
                   primary-class edge
                   eigenvalue1 edge
                   eigenvalue2 edge
                   slope edge
                   eigenvector1 edge
                   eigenvector2 edge)
                 (aref terrain-cell-map xcoord ycoord))))
    (t (let* ((z-matrix (math:transpose-matrix
                         (make-array '(1 9) :initial-contents
                              (list (mapcar 'feet-to-meter
                                   (car (get-tc-group terrain-cell-map
                                      xcoord ycoord 'elevation)))))))
          (k-matrix-u (math:multiply-matrices b-matrix-u z-matrix))
          (k-matrix (math:multiply-matrices norm-matrix k-matrix-u)))
       (setf (aref terrain-cell-map xcoord ycoord)
          (append (classify (calculate-eigenvalues-and-slope
                   (aref k-matrix 1)
                   (aref k-matrix 2)
                   (aref k-matrix 3)
                   (aref k-matrix 4)
                   (aref k-matrix 5)))
                  (aref terrain-cell-map xcoord ycoord)))
       (setf (aref terrain-cell-map xcoord ycoord)
          (append (calculate-gradient (aref k-matrix 1)(aref k-matrix 2))
                (aref terrain-cell-map xcoord ycoord)))))))))))



(defun calculate-eigenvalues-and-slope (k2 k3 k4 k5 k6)
 (let* ((a (+ (* 2 k4)(* 2 k6)))
      (b1 (- (* (- (* -2.0 k4)(* 2.0 k6))(- (* -2.0 k4) (* 2.0 k6)))
               (* 4.0 (- (* (* 2.0 k4)(* 2.0 k6))(* k5 k5))))))
      (cond ((< b1 0.0)
          (setq b1 0)))
  (let* ((b (sqrt b1))
      (eigenvalue1 (/ (+ a b) 2.0))
      (eigenvalue2 (/ (- a b) 2.0)))
```

```
  (setq slope (/ (* 360.0 (atan (sqrt (+ (* k2 k2)(* k3 k3))) 1.0)) 2.0 pi))
  (cond ((< (abs eigenvalue1)(abs eigenvalue2))
       (setq temp 0)
       (setq temp eigenvalue1)
       (setq eigenvalue1 eigenvalue2)
       (setq eigenvalue2 temp)))
  (let* ((B11 (- (* 2 k4) eigenvalue1))(number 0)(B12 (-  0 k5)))
   (setq normalized-eigenvector (sqrt (+ (* B12 B12)(* B11 B11))))
   (cond ((= normalized-eigenvector 0)
        (list 'eigenvalue1  (eval eigenvalue1)
            'eigenvalue2  (eval eigenvalue2)
            'slope        (eval slope)
            'eigenvector1 (eval number)
            'eigenvector2 (eval number)))
       (t (setq eigenvector1 (* (/ 1 normalized-eigenvector) B11))
          (setq eigenvector2 (* (/ 1 normalized-eigenvector) B12))
          (list 'eigenvalue1  (eval eigenvalue1)
              'eigenvalue2  (eval eigenvalue2)
              'slope        (eval slope)
              'eigenvector1 (eval eigenvector1)
              'eigenvector2 (eval eigenvector2)))))))


(defun classify (k)
 (let ((E1 (second k))(E2 (fourth k))(slope (sixth k)))
  (cond
   ((< slope slope-limit)
    (cond
     ((and (< E1 (- curv-threshold))
         (< E2 (- curv-threshold)))
      (append '(primary-class level secondary-class peak) k))
     ((and (> E1 curv-threshold)
         (> E2 curv-threshold))
      (append '(primary-class level secondary-class pit) k))
     ((and (< E1 (- curv-threshold))
         (< (abs E2) curv-threshold))
      (append '(primary-class level secondary-class ridge) k))
     ((and (> E1 curv-threshold)
         (< (abs E2) curv-threshold))
      (append '(primary-class level secondary-class ravine) k))
     ((and (< E1 (- curv-threshold))
         (> E2 curv-threshold))
      (append '(primary-class level secondary-class saddle) k))
     ((and (> E1 curv-threshold)
         (< E2 (- curv-threshold)))
```

```
        (append '(primary-class level secondary-class pass) k))
       (t
        (append '(primary-class level secondary-class flat) k))))
     ((> slope unsafe)
      (append '(primary-class unsafe-slope) k))
     (t
      (cond
       ((and (< E1 (- curv-threshold))
             (< E2 (- curv-threshold)))
        (append '(primary-class safe-slope secondary-class peak) k))
       ((and (> E1 curv-threshold)
             (> E2 curv-threshold))
        (append '(primary-class safe-slope secondary-class pit) k))
       ((and (< E1 (- curv-threshold))
             (< (abs E2) curv-threshold))
        (append '(primary-class safe-slope secondary-class ridge) k))
       ((and (> E1 curv-threshold)
             (< (abs E2) curv-threshold))
        (append '(primary-class safe-slope secondary-class ravine) k))
       ((and (< E1 (- curv-threshold))
             (> E2 curv-threshold))
        (append '(primary-class safe-slope secondary-class saddle) k))
       ((and (> E1 curv-threshold)
             (< E2 (- curv-threshold)))
        (append '(primary-class safe-slope secondary-class pass) k))
       (t
        (append '(primary-class safe-slope secondary-class planar) k)))))))


(defun calculate-gradient (k2 k3)
 (let ((slope (/ (* 360 (atan (sqrt (+ (* k2 k2)(* k3 k3))) 1.0)) 2.0 pi))(number 0))
  (cond
   ((> (abs slope) 0)
    (let ((gradient1 (* (/ 1 slope) k2))(gradient2 (* (/ 1 slope) k3)))
     (list 'gradient1 (eval gradient1)
           'gradient2 (eval gradient2))))
   (t
    (list 'gradient1 (eval number)
          'gradient2 (eval number))))))


(defun rline-test (terrain-cell-map)
 (let ((mapsize (sqrt (array-total-size terrain-cell-map))))
  (do ((xcoord 0 (1+ xcoord)))
      ((= xcoord mapsize))
```

```
(do ((ycoord 0 (1+ ycoord)))
  ((= ycoord mapsize))
 (cond
  ((edge-tc-p mapsize xcoord ycoord)
   (setf (aref terrain-cell-map xcoord ycoord)
       (append '(rline edge)
            (aref terrain-cell-map xcoord ycoord))))
   (t (let ((gradient1     (getf (aref terrain-cell-map xcoord ycoord)
                             'gradient1))
            (gradient2     (getf (aref terrain-cell-map xcoord ycoord)
                             'gradient2))
            (eigenvector1  (getf (aref terrain-cell-map xcoord ycoord)
                             'eigenvector1))
            (eigenvector2  (getf (aref terrain-cell-map xcoord ycoord)
                             'eigenvector2))
            (primary-class (getf (aref terrain-cell-map xcoord ycoord)
                             'primary-class))
            (eigenvalue1   (getf (aref terrain-cell-map xcoord ycoord)
                             'eigenvalue1)))
       (cond
      ((equal primary-class 'safe-slope)
       (cond
        ((< eigenvalue1 (- curv-threshold))
         (setq dot-product (abs (+ (* gradient2 eigenvector1)
                                   (* gradient1 eigenvector2))))
         (cond
          ((< dot-product dot-prod-threshold)
           (setf (aref terrain-cell-map xcoord ycoord)
               (append '(rline ridge)
                    (aref terrain-cell-map xcoord ycoord))))
          (t
           (setf (aref terrain-cell-map xcoord ycoord)
               (append '(rline no)
                    (aref terrain-cell-map xcoord ycoord))))))
        ((> eigenvalue1 curv-threshold)
         (setq dot-product (abs (+ (* gradient2 eigenvector1)
                                   (* gradient1 eigenvector2))))
          (cond
           ((< dot-product dot-prod-threshold)
            (setf (aref terrain-cell-map xcoord ycoord)
                (append '(rline ravine)
                     (aref terrain-cell-map xcoord ycoord))))
           (t
            (setf (aref terrain-cell-map xcoord ycoord)
                (append '(rline no)
```

```
                    (aref terrain-cell-map xcoord ycoord))))))))
            (t
             (setf (aref terrain-cell-map xcoord ycoord)
               (append '(rline no)
                    (aref terrain-cell-map xcoord ycoord))))))))))))


(defun write-classification (terrain-cell-map classfile)
 (setq output-stream (open classfile :direction :output))
 (let ((mapsize (sqrt (array-total-size terrain-cell-map))))
  (do ((ycoord 0 (1+ ycoord)))
     ((= ycoord mapsize))
    (do ((xcoord 0 (1+ xcoord)))
       ((= xcoord mapsize))
      (terpri output-stream)
      (let ((first-class  (getf (aref terrain-cell-map xcoord ycoord) 'primary-class))
          (second-class (getf (aref terrain-cell-map xcoord ycoord) 'secondary-class))
          (slope       (getf (aref terrain-cell-map xcoord ycoord) 'slope))
          (r-line      (getf (aref terrain-cell-map xcoord ycoord) 'rline)))
       (cond
        ((equal first-class 'safe-slope)
         (cond ((equal r-line      'ridge)  (prin1 '15 output-stream))
             ((equal r-line      'ravine) (prin1 '16 output-stream))
             ((equal second-class 'peak)   (prin1 '1 output-stream))
             ((equal second-class 'pit)    (prin1 '2 output-stream))
             ((equal second-class 'ridge)  (prin1 '9 output-stream))
             ((equal second-class 'ravine) (prin1 '11 output-stream))
             ((equal second-class 'saddle) (prin1 '10 output-stream))
             ((equal second-class 'pass)   (prin1 '12 output-stream))
             ((equal second-class 'planar)
              (cond ((> slope 25) (prin1 '47 output-stream))
                  ((> slope 21) (prin1 '46 output-stream))
                  ((> slope 17) (prin1 '45 output-stream))
                  ((> slope 13) (prin1 '44 output-stream))
                  ((> slope 9)  (prin1 '43 output-stream))
                  ((> slope 5)  (prin1 '42 output-stream))
                  (t          (prin1 '41 output-stream))))
             (t
              (prin1 '7 output-stream))))
        ((equal first-class 'level)
         (cond ((equal second-class 'peak)   (prin1 '1 output-stream))
             ((equal second-class 'pit)    (prin1 '2 output-stream))
             ((equal second-class 'ridge)  (prin1 '9 output-stream))
             ((equal second-class 'ravine) (prin1 '11 output-stream))
             ((equal second-class 'saddle) (prin1 '10 output-stream))
```

```
        ((equal second-class 'pass)  (prin1 '12 output-stream))
        ((equal second-class 'flat)  (prin1 '3 output-stream))
        (t
         (prin1 '8 output-stream))))
      ((equal first-class 'edge)       (prin1 '5 output-stream))
      ((equal first-class 'unsafe-slope)(prin1 '6 output-stream)))))))
      (close output-stream))


(defun get-tc (terrain-cell-map xcoord ycoord)
 (let ((mapsize (sqrt (array-total-size terrain-cell-map))))
  (cond
   ((range-tc-p mapsize xcoord ycoord)
    (aref terrain-cell-map xcoord ycoord)))))


(defun get-tc-group (terrain-cell-map xcoord ycoord slot)
 (let ((mapsize (sqrt (array-total-size terrain-cell-map))))
  (cond
   ((and (not (edge-tc-p mapsize xcoord ycoord))
         (range-tc-p mapsize xcoord ycoord))
    (list
     (cons (getf (aref terrain-cell-map (1- xcoord) (1+ ycoord)) slot)
     (cons (getf (aref terrain-cell-map    xcoord  (1+ ycoord)) slot)
     (cons (getf (aref terrain-cell-map (1+ xcoord) (1+ ycoord)) slot)
     (cons (getf (aref terrain-cell-map (1- xcoord)    ycoord ) slot)
     (cons (getf (aref terrain-cell-map    xcoord     ycoord ) slot)
     (cons (getf (aref terrain-cell-map (1+ xcoord)    ycoord ) slot)
     (cons (getf (aref terrain-cell-map (1- xcoord) (1- ycoord)) slot)
     (cons (getf (aref terrain-cell-map    xcoord  (1- ycoord)) slot)
     (cons (getf (aref terrain-cell-map (1+ xcoord) (1- ycoord)) slot)
         nil)))))))))))))


(defun edge-tc-p (mapsize xcoord ycoord)
 (or (= xcoord 0)
    (= ycoord 0)
    (= xcoord (1- mapsize))
    (= ycoord (1- mapsize))))



(defun corner-tc-p (mapsize xcoord ycoord)
 (or (and (= xcoord 0) (= ycoord 0))
    (and (= xcoord 0) (= ycoord (1- mapsize)))
```

```
    (and (= ycoord 0) (= xcoord (1- mapsize)))
    (and (= ycoord (1- mapsize))
        (= xcoord (1- mapsize)))))


(defun range-tc-p (mapsize xcoord ycoord)
  (and (and (>= xcoord 0) (< xcoord mapsize))
      (and (>= ycoord 0) (< ycoord mapsize))))
```

APPENDIX D - SLOPE CLASSIFICATION FUNCTIONS

```
;;; -*- Mode: LISP; Syntax: Common-lisp; Base: 10 -*-

;*****************************************************************
;*                  slope-utilities.lisp
;*****************************************************************
;*****************************************************************
;* List of functions in this file
;*
;* 1. run-slope - Top level function to run the slope program. Inputs
;*    are mapsize which is the size of the array holding the data,
;*    (typically 80). The datafile is the input file of elevation
;*    data. The slopefile is the output file of slope codes
;*    determined by the function. A sample call of the function
;*    would be (run-slope 80 "fq5886.1-hr-e" "fq5886.slope").
;*    Most of the calls in this function are identical to the calls
;*    in the run-class function. However, this function has been
;*    written separately since it is generally not run as often.
;* 2. write-slope - This function outputs a code to the slopefile
;*    for each array element, for later ease of display.
;*****************************************************************
;*****************************************************************


(defun run-slope (mapsize datafile slopefile)
 (make-tc-map hunter mapsize)
 (load-tc-attribute hunter datafile 'elevation)
 (load-tc-map hunter)
 (write-slope hunter slopefile)
 (send *blue-window* :refresh)
 (display-slope mapsize slopefile 100 100 10))


(defun write-slope (terrain-cell-map slopefile)
 (setq output-stream (open slopefile :direction :output))
 (let ((mapsize (sqrt (array-total-size terrain-cell-map))))
  (do ((ycoord 0 (1+ ycoord)))
      ((= ycoord mapsize))
    (do ((xcoord 0 (1+ xcoord)))
        ((= xcoord mapsize))
      (terpri output-stream)
      (let
       ((first-class (getf (aref terrain-cell-map xcoord ycoord) 'primary-class))
```

```
      (slope      (getf (aref terrain-cell-map xcoord ycoord) 'slope)))
      (cond
       ((equal first-class 'level)       (prin1 '3 output-stream))
       ((equal first-class 'safe-slope)
        (cond  ((> slope 25) (prin1 '27 output-stream))
               ((> slope 21) (prin1 '26 output-stream))
               ((> slope 17) (prin1 '25 output-stream))
               ((> slope 13) (prin1 '24 output-stream))
               ((> slope 9)  (prin1 '23 output-stream))
               ((> slope 5)  (prin1 '22 output-stream))
               (t          (prin1 '21 output-stream))))
       ((equal first-class 'edge)        (prin1 '5 output-stream))
       ((equal first-class 'unsafe-slope) (prin1 '6 output-stream)))))))
(close output-stream))
```

APPENDIX E - CONTOUR CLASSIFICATION FUNCTIONS

```
;;; -*- Mode: LISP; Syntax: Common-lisp; Base: 10 -*-

;*****************************************************************
;*                contour-utilities.lisp
;*****************************************************************
;*****************************************************************
;* List of functions in this file
;*
;* 1. run-contour - This is the top level function to run the contour
;*    program. The inputs are mapsize which is the size of the array
;*    holding elevation data (typically 80). Datafile which is the
;*    input file containing the elevation data, and contourfile which
;*    is the output file for contour codes based on the elevation.
;*    Sample call (run-contour 80 "fq5886.1-nr-e" "fq5886.cont").
;* 2. contour-map - This function puts a code into the current array
;*    location based on the contour interval in which the present cell
;*    is located.
;* 3. write-contour - This function outputs a code to the contour file
;*    for each array element for later ease of display.
;*****************************************************************
;*****************************************************************


(defun run-contour (mapsize datafile contourfile)
 (make-tc-map hunter mapsize)
 (load-tc-attribute hunter datafile 'elevation)
 (contour-map hunter)
 (write-contour hunter contourfile)
 (display-cont mapsize contourfile 100 100 10))



(defun contour-map (terrain-cell-map)
 (let ((mapsize (sqrt (array-total-size terrain-cell-map))))
  (do ((xcoord 0 (1+ xcoord)))
     ((= xcoord mapsize))
    (do ((ycoord 0 (1+ ycoord)))
       ((= ycoord mapsize))
      (let ((elevation (getf (aref terrain-cell-map xcoord ycoord) 'elevation)))
       (cond
         ((< elevation 1000)(setf (aref terrain-cell-map xcoord ycoord)
              (append '(contour 0)  (aref terrain-cell-map xcoord ycoord))))
```

```
((and (>= elevation 1000)(< elevation 1040))
  (setf (aref terrain-cell-map xcoord ycoord)
    (append '(contour 1) (aref terrain-cell-map xcoord ycoord))))
((and (>= elevation 1040)(< elevation 1080))
  (setf (aref terrain-cell-map xcoord ycoord)
    (append '(contour 2) (aref terrain-cell-map xcoord ycoord))))
((and (>= elevation 1080)(< elevation 1120))
  (setf (aref terrain-cell-map xcoord ycoord)
    (append '(contour 3) (aref terrain-cell-map xcoord ycoord))))
((and (>= elevation 1120)(< elevation 1160))
  (setf (aref terrain-cell-map xcoord ycoord)
    (append '(contour 4) (aref terrain-cell-map xcoord ycoord))))
((and (>= elevation 1160)(< elevation 1200))
  (setf (aref terrain-cell-map xcoord ycoord)
    (append '(contour 5) (aref terrain-cell-map xcoord ycoord))))
((and (>= elevation 1200)(< elevation 1240))
  (setf (aref terrain-cell-map xcoord ycoord)
    (append '(contour 6) (aref terrain-cell-map xcoord ycoord))))
((and (>= elevation 1240)(< elevation 1280))
  (setf (aref terrain-cell-map xcoord ycoord)
    (append '(contour 7) (aref terrain-cell-map xcoord ycoord))))
((and (>= elevation 1280)(< elevation 1320))
  (setf (aref terrain-cell-map xcoord ycoord)
    (append '(contour 8) (aref terrain-cell-map xcoord ycoord))))
((and (>= elevation 1320)(< elevation 1360))
  (setf (aref terrain-cell-map xcoord ycoord)
    (append '(contour 9) (aref terrain-cell-map xcoord ycoord))))
((and (>= elevation 1360)(< elevation 1400))
  (setf (aref terrain-cell-map xcoord ycoord)
    (append '(contour 10) (aref terrain-cell-map xcoord ycoord))))
((and (>= elevation 1400)(< elevation 1440))
  (setf (aref terrain-cell-map xcoord ycoord)
    (append '(contour 11) (aref terrain-cell-map xcoord ycoord))))
((and (>= elevation 1440)(< elevation 1480))
  (setf (aref terrain-cell-map xcoord ycoord)
    (append '(contour 12) (aref terrain-cell-map xcoord ycoord))))
((and (>= elevation 1480)(< elevation 1520))
  (setf (aref terrain-cell-map xcoord ycoord)
    (append '(contour 13) (aref terrain-cell-map xcoord ycoord))))
((and (>= elevation 1520)(< elevation 1560))
  (setf (aref terrain-cell-map xcoord ycoord)
    (append '(contour 14) (aref terrain-cell-map xcoord ycoord))))
((and (>= elevation 1560)(< elevation 1600))
  (setf (aref terrain-cell-map xcoord ycoord)
    (append '(contour 15) (aref terrain-cell-map xcoord ycoord))))
```

```
        ((and (>= elevation 1600)(< elevation 1640))
          (setf (aref terrain-cell-map xcoord ycoord)
            (append '(contour 16) (aref terrain-cell-map xcoord ycoord))))
        ((and (>= elevation 1640)(< elevation 1680))
          (setf (aref terrain-cell-map xcoord ycoord)
            (append '(contour 17) (aref terrain-cell-map xcoord ycoord))))
        ((and (>= elevation 1680)(< elevation 1720))
          (setf (aref terrain-cell-map xcoord ycoord)
            (append '(contour 18) (aref terrain-cell-map xcoord ycoord))))
        ((and (>= elevation 1720)(< elevation 1760))
          (setf (aref terrain-cell-map xcoord ycoord)
            (append '(contour 19) (aref terrain-cell-map xcoord ycoord))))
        ((and (>= elevation 1760)(< elevation 1800))
          (setf (aref terrain-cell-map xcoord ycoord)
            (append '(contour 20) (aref terrain-cell-map xcoord ycoord))))
        ((>= elevation 1800)
          (setf (aref terrain-cell-map xcoord ycoord)
            (append '(contour 21) (aref terrain-cell-map xcoord ycoord))))))))))


(defun write-contour (terrain-cell-map contourfile)
  (setq output-stream (open contourfile :direction :output))
  (let ((mapsize (sqrt (array-total-size terrain-cell-map))))
    (do ((ycoord 0 (1+ ycoord)))
        ((= ycoord mapsize))
        (do ((xcoord 0 (1+ xcoord)))
            ((= xcoord mapsize))
        (terpri output-stream)
        (let ((contour (getf (aref terrain-cell-map xcoord ycoord) 'contour)))
          (cond
            ((= contour 0) (prin1 '0 output-stream))
            ((= contour 1) (prin1 '1 output-stream))
            ((= contour 2) (prin1 '2 output-stream))
            ((= contour 3) (prin1 '3 output-stream))
            ((= contour 4) (prin1 '4 output-stream))
            ((= contour 5) (prin1 '5 output-stream))
            ((= contour 6) (prin1 '6 output-stream))
            ((= contour 7) (prin1 '7 output-stream))
            ((= contour 8) (prin1 '8 output-stream))
            ((= contour 9) (prin1 '9 output-stream))
            ((= contour 10)(prin1 '10 output-stream))
            ((= contour 11)(prin1 '11 output-stream))
            ((= contour 12)(prin1 '12 output-stream))
            ((= contour 13)(prin1 '13 output-stream))
```

```
        ((= contour 14)(prin1 '14 output-stream))
        ((= contour 15)(prin1 '15 output-stream))
        ((= contour 16)(prin1 '16 output-stream))
        ((= contour 17)(prin1 '17 output-stream))
        ((= contour 18)(prin1 '18 output-stream))
        ((= contour 19)(prin1 '19 output-stream))
        ((= contour 20)(prin1 '20 output-stream))
        ((= contour 21)(prin1 '21 output-stream)))))))
(close output-stream))
```

APPENDIX F - B-SPLINE SMOOTHING FUNCTIONS

```
;;; -*- Mode: LISP; Syntax: Common-lisp; Base: 10 -*-

;*****************************************************************
;*                b-spline-utilities.lisp
;*****************************************************************
;*****************************************************************
;* List of Functions in this file
;*
;* 1. There are seven variables declared at the beginning of this
;*    file.  These are the s and t matrices which are varied from
;*    zero to one over the patch to calculate elevation at a point.
;*    The transpose of both t matrices is required in the calculation.
;*    Additionally, the geom-matrix-z has been initialized.
;* 2. run-smooth-elev - This is the top level function to smooth
;*    the elevation data using b-splines.  The inputs are mapsize
;*    which is the size of the array holding the elevation data
;*    (for a typical high resolution grid square the array would be
;*    80 x 80, so mapsize would be 80).  The datafile is the input
;*    file of elevation data (in double quotes), and the elevfile
;*    is the output file of smoothed data (also in double quotes).
;*    A sample call of this function would look like the following:
;*    (run-smooth-elev 80 "fq5886.1-hr-e" "fq5886.1-hr-e-sm")
;* 3. calculate-smooth-elev - This controls the principle loop in
;*    the function to pass through each element of the array and
;*    determine the smooth elevation.  It sets the smooth elevation
;*    to the present elevation if it is an edge cell.  It calls on
;*    three other functions if it is an interior cell or on the
;*    right or bottom sides of the array.
;* 4. right-side - Calls the actual smoothing function with the
;*    appropriate parameters if the current cell is on the right of
;*    the grid square against the edge.
;* 5. bottom-side - Calls the actual smoothing function with the
;*    appropriate parameters if the current cell is on the bottom of
;*    the grid square against the edge.
;* 6. interior-of-array - Handles the majority of the cells.  Calls
;*    actual smoothing function with the appropriate parameters
;*    for the typical cell that is not a boundary condition.
;* 7. smooth-elev - Primary mathematical function that returns the
;*    smoothed elevation at the requested point after the necessary
;*    matrix manipulations.
;* 8. load-geom-matrix-z - This function loads the geometry matrix
;*    with the elevation data.  The sixteen points loaded are the
```

```
;*     control points in the z direction which determine the shape
;*     of the surface patch.
;* 9. write-smooth-elev - This function outputs the smoothed elevations
;*     from the array to the elevfile designated by the user.  This
;*     new elevfile will subsequently be input for the main function
;*     of classifying terrain.
;*****************************************************************
;*****************************************************************


(setq s-zero-matrix (make-array '(1 4) :initial-contents
                        '((0 0 0 1))))

(setq s-one-matrix  (make-array '(1 4) :initial-contents
                        '((1 1 1 1))))

(setq t-zero-matrix (make-array '(1 4) :initial-contents
                        '((0 0 0 1))))

(setq t-one-matrix  (make-array '(1 4) :initial-contents
                        '((1 1 1 1))))

(setq t-zero-matrix-t (math:transpose-matrix t-zero-matrix))

(setq t-one-matrix-t  (math:transpose-matrix t-one-matrix))

(setq geom-matrix-z (make-array '(4 4) :initial-contents
                        '((0 0 0 0)
                          (0 0 0 0)
                          (0 0 0 0)
                          (0 0 0 0))))


(defun run-smooth-elev (mapsize datafile elevfile)
  (make-tc-map hunter mapsize)
  (load-tc-attribute hunter datafile 'elevation)
  (calculate-smooth-elev hunter)
  (write-smooth-elev hunter elevfile))


(defun calculate-smooth-elev (terrain-cell-map)
  (let ((mapsize (sqrt (array-total-size terrain-cell-map))))
    (do ((xcoord 0 (1+ xcoord)))
        ((= xcoord mapsize))
      (do ((ycoord 0 (1+ ycoord)))
```

```
      ((= ycoord mapsize))
     (cond
      ((edge-tc-p mapsize xcoord ycoord)
       (let ((elevation (getf (aref terrain-cell-map xcoord ycoord) 'elevation)))
        (setf (aref terrain-cell-map xcoord ycoord)
            (append (list 'smooth-elevation (eval elevation))
             (aref terrain-cell-map xcoord ycoord)))))
      ((= xcoord (- mapsize 2))
       (right-side terrain-cell-map mapsize xcoord ycoord))
      ((= ycoord (- mapsize 2))
       (bottom-side terrain-cell-map xcoord ycoord))
      (t (interior-of-array terrain-cell-map xcoord ycoord)))))))


(defun right-side (terrain-cell-map mapsize xcoord ycoord)
 (cond
  ((= ycoord (- mapsize 2))
   (setf (aref terrain-cell-map xcoord ycoord)
    (append (list 'smooth-elevation (smooth-elev terrain-cell-map
                (1- xcoord) (1- ycoord) s-zero-matrix t-one-matrix-t))
           (aref terrain-cell-map xcoord ycoord))))
  (t
   (setf (aref terrain-cell-map xcoord ycoord)
    (append (list 'smooth-elevation (smooth-elev terrain-cell-map
                (1- xcoord) ycoord s-one-matrix t-one-matrix-t))
           (aref terrain-cell-map xcoord ycoord))))))


(defun bottom-side (terrain-cell-map xcoord ycoord)
 (setf (aref terrain-cell-map xcoord ycoord)
     (append (list 'smooth-elevation (smooth-elev terrain-cell-map
                    xcoord (1- ycoord) s-zero-matrix t-zero-matrix-t))
           (aref terrain-cell-map xcoord ycoord))))


(defun interior-of-array (terrain-cell-map xcoord ycoord)
 (setf (aref terrain-cell-map xcoord ycoord)
     (append (list 'smooth-elevation (smooth-elev terrain-cell-map
                    xcoord ycoord s-one-matrix t-zero-matrix-t))
           (aref terrain-cell-map xcoord ycoord))))


(defun smooth-elev (terrain-cell-map xcoord ycoord s-matrix t-matrix)
 (load-geom-matrix-z terrain-cell-map xcoord ycoord 'elevation)
 (setq a (math:multiply-matrices s-matrix b-spline-matrix))
```

```
(setq b (math:multiply-matrices a geom-matrix-z))
(setq c (math:multiply-matrices b b-spline-matrix-t))
(setq d (math:multiply-matrices c t-matrix))
(setq e (* 1.0 (aref d 0))))



(defun load-geom-matrix-z (terrain-cell-map xcoord ycoord slot)
 (let ((mapsize (sqrt (array-total-size terrain-cell-map))))
  (cond
   ((and (not (edge-tc-p mapsize xcoord ycoord))
         (range-tc-p mapsize xcoord ycoord))
    (setf (aref geom-matrix-z 3 0)
          (getf (aref terrain-cell-map (1- xcoord)   (1- ycoord  )) slot))
    (setf (aref geom-matrix-z 3 1)
          (getf (aref terrain-cell-map    xcoord    (1- ycoord  )) slot))
    (setf (aref geom-matrix-z 3 2)
          (getf (aref terrain-cell-map (1+ xcoord)   (1- ycoord  )) slot))
    (setf (aref geom-matrix-z 3 3)
          (getf (aref terrain-cell-map (+  xcoord 2) (1- ycoord  )) slot))
    (setf (aref geom-matrix-z 2 0)
          (getf (aref terrain-cell-map (1- xcoord)      ycoord   ) slot))
    (setf (aref geom-matrix-z 2 1)
          (getf (aref terrain-cell-map    xcoord        ycoord   ) slot))
    (setf (aref geom-matrix-z 2 2)
          (getf (aref terrain-cell-map (1+ xcoord)      ycoord   ) slot))
    (setf (aref geom-matrix-z 2 3)
          (getf (aref terrain-cell-map (+  xcoord 2)    ycoord   ) slot))
    (setf (aref geom-matrix-z 1 0)
          (getf (aref terrain-cell-map (1- xcoord)   (1+ ycoord  )) slot))
    (setf (aref geom-matrix-z 1 1)
          (getf (aref terrain-cell-map    xcoord    (1+ ycoord  )) slot))
    (setf (aref geom-matrix-z 1 2)
          (getf (aref terrain-cell-map (1+ xcoord)   (1+ ycoord  )) slot))
    (setf (aref geom-matrix-z 1 3)
          (getf (aref terrain-cell-map (+  xcoord 2) (1+ ycoord  )) slot))
    (setf (aref geom-matrix-z 0 0)
          (getf (aref terrain-cell-map (1- xcoord)   (+  ycoord 2)) slot))
    (setf (aref geom-matrix-z 0 1)
          (getf (aref terrain-cell-map    xcoord    (+  ycoord 2)) slot))
    (setf (aref geom-matrix-z 0 2)
          (getf (aref terrain-cell-map (1+ xcoord)   (+  ycoord 2)) slot))
    (setf (aref geom-matrix-z 0 3)
          (getf (aref terrain-cell-map (+  xcoord 2) (+  ycoord 2)) slot))))))
```

```
(defun write-smooth-elev (terrain-cell-map elevfile)
 (setq output-stream (open elevfile :direction :output))
 (let ((mapsize (sqrt (array-total-size terrain-cell-map))))
  (do ((xcoord 0 (1+ xcoord)))
      ((= xcoord mapsize))
    (do ((ycoord (1- mapsize) (1- ycoord)))
        ((< ycoord 0))
     (terpri output-stream)
     (let ((elevation (getf (aref terrain-cell-map xcoord ycoord) 'smooth-elevation)))
      (prin1 elevation output-stream)))))
 (close output-stream))
```

APPENDIX G - GRAPHICS DRAWING FUNCTIONS

```
;;; -*- Mode: LISP; Syntax: Common-lisp; Base: 10 -*-

;*****************************************************************
;*                    draw-utilities.lisp
;*****************************************************************
;*****************************************************************
;* List of functions in this file
;*
;* 1. The initial portion of this file contains the functions to
;*    declare the color window on the color monitor and a number of
;*    variable color definitions necessary for the associated displays
;*    of information.
;*
;* 2. box - This draws a box in the specified color at the location
;*    indicated.  This is called repeatedly by almost all the functions.
;* 3. display-class - This function displays the classifications
;*    calculated by previous functions. The inputs are mapsize, which
;*    is typically 80, classfile which is the file holding the data
;*    calculated by run-class, x-start which is the x coordinate at
;*    which to begin the drawing, y-start which is the y coordinate
;*    at which to begin the drawing, and scale which is the size for
;*    each square of the display. The scale can vary from 1 to 12, but
;*    the best results are in the range 6 to 11.  A sample call of this
;*    function would be (display-class 80 "fq5886.class" 100 100 10),
;*    this would draw the scale display near the middle of the
;*    monitor with a scale size of 10.  To get four displays on the
;*    same screen, a scale size of 6 is required.  All the other
;*    display functions work the same as this on as far as parameters.
;* 4. display-slope - Displays the slope information on the monitor.
;* 5. display-cont - Displays the contour information on the monitor.
;* 6. display-veg - Displays the vegetation data on the monitor.
;* 7. display-scale-class - Draws a color scale for the class display
;*    depicting the meaning of each color.
;* 8. display-scale-slope - Draws a color scale for the slope display
;*    depicting the meaning of each color.
;* 9. display-scale-cont - Draws a color scale for the contour display
;*    depicting the meaning of each color.
;* 10. display-scale-veg - Draws a color scale for the vegetation display
;*    depicting the meaning of each color.
;*
;* 11. The various functions at the end of the file have been written
;*    to facilitate displaying results on the monitor.  Each is merely
```

```
;*     a number of calls to various functions to display calculated
;*     information on a single screen.  Additional functions could
;*     be easily added to display information best suited to the user's
;*     needs or desires.
;*****************************************************************
;*****************************************************************


(defun make-color-window
      (&rest options &key (superior (color:find-color-screen : create-p t))
      &allow-other-keys)
  (apply #'tv:make-window 'tv:window
       :blinker-p nil
       :borders 2
       :save-bits nil
       :expose-p t
       :default-character-style
         '(:fix :bold :large)
       :label "  FORT HUNTER LIGGETT TERRAIN DATA"
       :superior superior
       options))


(defvar *color-window* (make-color-window))

(defun make-blue-window (&rest options)
  (apply #'make-color-window
       :erase-aluf(send color:color-screen
                    :compute-color-alu
                    tv:alu-seta 0 0 0.5)
       options))

(defvar *blue-window* (make-blue-window))


;   Typical color variables

(defvar *red-alu*    (send(send *blue-window* :screen)
                :compute-color-alu color:alu-x 0.9 0 0))
(defvar *orange-alu* (send(send *blue-window* :screen)
                :compute-color-alu color:alu-x 1.0 0.6 0.0))
(defvar *orange1-alu* (send(send *blue-window* :screen)
                :compute-color-alu color:alu-x 0.8 0.4 0.0))
(defvar *yellow-alu* (send(send *blue-window* :screen)
                :compute-color-alu color:alu-x 1.0 .93 0.2))
```

```
(defvar *yellow1-alu* (send(send *blue-window* :screen)
                :compute-color-alu color:alu-x 0.9 .9 0.3))
(defvar *blue-alu*   (send(send *blue-window* :screen)
                :compute-color-alu color:alu-x 0 0 0.6))
(defvar *blue1-alu*  (send(send *blue-window* :screen)
                :compute-color-alu color:alu-x 0.3 0.3 0.8))
(defvar *blue2-alu*  (send(send *blue-window* :screen)
                :compute-color-alu color:alu-x 0.5 0.5 1.0))
(defvar *green-alu*  (send(send *blue-window* :screen)
                :compute-color-alu color:alu-x 0 0.9 0))
(defvar *black-alu*  (send(send *blue-window* :screen)
                :compute-color-alu color:alu-x 0 0 0))
(defvar *black1-alu* (send(send *blue-window* :screen)
                :compute-color-alu color:alu-x 0.4 0.4 0.4))
(defvar *black2-alu* (send(send *blue-window* :screen)
                :compute-color-alu color:alu-x 0.6 0.6 0.6))
(defvar *tan-alu*    (send(send *blue-window* :screen)
                :compute-color-alu color:alu-x 1.0 0.9 0.5))
(defvar *blugre-alu* (send(send *blue-window* :screen)
                :compute-color-alu color:alu-x 0.0 0.6 0.6))
(defvar *purple-alu* (send(send *blue-window* :screen)
                :compute-color-alu color:alu-x 0.6 0.0 0.6))
(defvar *white-alu*  (send(send *blue-window* :screen)
                :compute-color-alu color:alu-x 1.0 1.0 1.0))


;     Variables for the slope color ramp

(defvar *yel1-alu*  (send(send *blue-window* :screen)
                :compute-color-alu color:alu-x 1.0 1.0 0.70))
(defvar *yel2-alu*  (send(send *blue-window* :screen)
                :compute-color-alu color:alu-x 1.0 1.0 0.63))
(defvar *yel3-alu*  (send(send *blue-window* :screen)
                :compute-color-alu color:alu-x 1.0 1.0 0.55))
(defvar *yel4-alu*  (send(send *blue-window* :screen)
                :compute-color-alu color:alu-x 1.0 1.0 0.45))
(defvar *yel5-alu*  (send(send *blue-window* :screen)
                :compute-color-alu color:alu-x 1.0 1.0 0.30))
(defvar *yel6-alu*  (send(send *blue-window* :screen)
                :compute-color-alu color:alu-x 1.0 1.0 0.15))
(defvar *yel7-alu*  (send(send *blue-window* :screen)
                :compute-color-alu color:alu-x 1.0 1.0 0.0))


;     Variables for the elevation color ramp
```

```
(defvar *red9-alu*   (send(send *blue-window* :screen)
                :compute-color-alu color:alu-x 0.4 0 0))
(defvar *red8-alu*   (send(send *blue-window* :screen)
                :compute-color-alu color:alu-x 0.55 0.2 0.2))
(defvar *red7-alu*   (send(send *blue-window* :screen)
                :compute-color-alu color:alu-x 0.7 0.4 0.4))
(defvar *red6-alu*   (send(send *blue-window* :screen)
                :compute-color-alu color:alu-x 0.85 0.6 0.6))
(defvar *red5-alu*   (send(send *blue-window* :screen)
                :compute-color-alu color:alu-x 1.0 0.8 0.8))
(defvar *green9-alu* (send(send *blue-window* :screen)
                :compute-color-alu color:alu-x 0 0.3 0))
(defvar *green8-alu* (send(send *blue-window* :screen)
                :compute-color-alu color:alu-x 0.12 0.45 0.12))
(defvar *green7-alu* (send(send *blue-window* :screen)
                :compute-color-alu color:alu-x 0.24 0.6 0.24))
(defvar *green6-alu* (send(send *blue-window* :screen)
                :compute-color-alu color:alu-x 0.36 0.75 0.36))
(defvar *green5-alu* (send(send *blue-window* :screen)
                :compute-color-alu color:alu-x 0.48 1.0 0.48))
(defvar *blue9-alu*  (send(send *blue-window* :screen)
                :compute-color-alu color:alu-x 0 0 0.4))
(defvar *blue8-alu*  (send(send *blue-window* :screen)
                :compute-color-alu color:alu-x 0.2 0.2 0.55))
(defvar *blue7-alu*  (send(send *blue-window* :screen)
                :compute-color-alu color:alu-x 0.4 0.4 0.7))
(defvar *blue6-alu*  (send(send *blue-window* :screen)
                :compute-color-alu color:alu-x 0.6 0.6 0.85))
(defvar *blue5-alu*  (send(send *blue-window* :screen)
                :compute-color-alu color:alu-x 0.8 0.8 1.0))
(defvar *yellow9-alu* (send(send *blue-window* :screen)
                :compute-color-alu color:alu-x .3 .3 0.1))
(defvar *yellow8-alu* (send(send *blue-window* :screen)
                :compute-color-alu color:alu-x .45 .45 0.2))
(defvar *yellow7-alu* (send(send *blue-window* :screen)
                :compute-color-alu color:alu-x .6 .6 0.3))
(defvar *yellow6-alu* (send(send *blue-window* :screen)
                :compute-color-alu color:alu-x .75 .75 0.4))
(defvar *yellow5-alu* (send(send *blue-window* :screen)
                :compute-color-alu color:alu-x 0.9 0.9 0.5))
(defvar *orange9-alu* (send(send *blue-window* :screen)
                :compute-color-alu color:alu-x 1.0 0.4 0))
```

```
;   Drawing functions for the various algorithms

(defun box (width height xcoord ycoord shade)
 (send *blue-window* :draw-rectangle width height xcoord ycoord shade))


(defun display-class (mapsize classfile x-start y-start scale)
 (setq input-stream (open classfile :direction :input))
 (box (+ (* scale mapsize) 20) (+ (* scale mapsize) 20) x-start y-start *black-alu*)
 (cond ((and (= mapsize 80) (> scale 5))
       (display-scale-class (+ (+ (* scale mapsize) 20) x-start) y-start scale
                 (+ 10 (* scale 18)) (+ (* scale mapsize) 20))))
 (do ((ycoord (+ y-start 10) (+ ycoord scale)))
    ((= ycoord (+ (* scale mapsize) (+ y-start 10))))
  (do ((xcoord (+ x-start 10) (+ xcoord scale)))
     ((= xcoord (+ (* scale mapsize) (+ x-start 10))))
    (let ((class (read input-stream)))
     (cond
      ((= class 1) (box scale scale xcoord ycoord *black-alu*))
      ((= class 2) (box scale scale xcoord ycoord *blue-alu*))
      ((= class 3) (box scale scale xcoord ycoord *green-alu*))
      ((= class 41)(box scale scale xcoord ycoord *yel1-alu*))
      ((= class 42)(box scale scale xcoord ycoord *yel2-alu*))
      ((= class 43)(box scale scale xcoord ycoord *yel3-alu*))
      ((= class 44)(box scale scale xcoord ycoord *yel4-alu*))
      ((= class 45)(box scale scale xcoord ycoord *yel5-alu*))
      ((= class 46)(box scale scale xcoord ycoord *yel6-alu*))
      ((= class 47)(box scale scale xcoord ycoord *yel7-alu*))
      ((= class 5) (box scale scale xcoord ycoord *red8-alu*))
      ((= class 6) (box scale scale xcoord ycoord *red-alu*))
      ((= class 7) (box scale scale xcoord ycoord *white-alu*))
      ((= class 8) (box scale scale xcoord ycoord *tan-alu*))
      ((= class 9) (box scale scale xcoord ycoord *black1-alu*))
      ((= class 10)(box scale scale xcoord ycoord *black2-alu*))
      ((= class 11)(box scale scale xcoord ycoord *blue1-alu*))
      ((= class 12)(box scale scale xcoord ycoord *blue2-alu*))
      ((= class 15)(box scale scale xcoord ycoord *purple-alu*))
      ((= class 16)(box scale scale xcoord ycoord *blugre-alu*))))))
 (close input-stream))


(defun display-slope (mapsize classfile x-start y-start scale)
 (setq input-stream (open classfile :direction :input))
 (box (+ (* scale mapsize) 20) (+ (* scale mapsize) 20) x-start y-start *black-alu*)
 (cond ((and (= mapsize 80) (> scale 5))
```

```
      (display-scale-slope (+ (+ (* scale mapsize) 20) x-start) y-start scale
                (+ 10 (* scale 18)) (+ (* scale mapsize) 20))))
 (do ((ycoord (+ y-start 10) (+ ycoord scale)))
    ((= ycoord (+ (* scale mapsize) (+ y-start 10))))
   (do ((xcoord (+ x-start 10) (+ xcoord scale)))
      ((= xcoord (+ (* scale mapsize) (+ x-start 10))))
    (let ((slope (read input-stream)))
     (cond
      ((= slope 21) (box scale scale xcoord ycoord *yel1-alu*))
      ((= slope 22) (box scale scale xcoord ycoord *yel2-alu*))
      ((= slope 23) (box scale scale xcoord ycoord *yel3-alu*))
      ((= slope 24) (box scale scale xcoord ycoord *yel4-alu*))
      ((= slope 25) (box scale scale xcoord ycoord *yel5-alu*))
      ((= slope 26) (box scale scale xcoord ycoord *yel6-alu*))
      ((= slope 27) (box scale scale xcoord ycoord *yel7-alu*))
      ((= slope 3)  (box scale scale xcoord ycoord *green-alu*))
      ((= slope 5)  (box scale scale xcoord ycoord *red8-alu*))
      ((= slope 6)  (box scale scale xcoord ycoord *red-alu*))))))
 (close input-stream))


(defun display-cont (mapsize classfile x-start y-start scale)
 (setq input-stream (open classfile :direction :input))
 (box (+ (* scale mapsize) 20) (+ (* scale mapsize) 20) x-start y-start *black-alu*)
 (cond ((and (= mapsize 80) (> scale 5))
      (display-scale-cont (+ (+ (* scale mapsize) 20) x-start) y-start scale
                (+ 10 (* scale 18)) (+ (* scale mapsize) 20))))
 (do ((ycoord (+ y-start 10) (+ ycoord scale)))
    ((= ycoord (+ (* scale mapsize) (+ y-start 10))))
   (do ((xcoord (+ x-start 10) (+ xcoord scale)))
      ((= xcoord (+ (* scale mapsize) (+ x-start 10))))
    (let ((contour (read input-stream)))
     (cond
      ((= contour 0) (box scale scale xcoord ycoord *red-alu*))
      ((= contour 1) (box scale scale xcoord ycoord *blue9-alu*))
      ((= contour 2) (box scale scale xcoord ycoord *blue8-alu*))
      ((= contour 3) (box scale scale xcoord ycoord *blue7-alu*))
      ((= contour 4) (box scale scale xcoord ycoord *blue6-alu*))
      ((= contour 5) (box scale scale xcoord ycoord *blue5-alu*))
      ((= contour 6) (box scale scale xcoord ycoord *green9-alu*))
      ((= contour 7) (box scale scale xcoord ycoord *green8-alu*))
      ((= contour 8) (box scale scale xcoord ycoord *green7-alu*))
      ((= contour 9) (box scale scale xcoord ycoord *green6-alu*))
      ((= contour 10)(box scale scale xcoord ycoord *green5-alu*))
      ((= contour 11)(box scale scale xcoord ycoord *red9-alu*))
```

```
        ((= contour 12)(box scale scale xcoord ycoord *red8-alu*))
        ((= contour 13)(box scale scale xcoord ycoord *red7-alu*))
        ((= contour 14)(box scale scale xcoord ycoord *red6-alu*))
        ((= contour 15)(box scale scale xcoord ycoord *red5-alu*))
        ((= contour 16)(box scale scale xcoord ycoord *yellow9-alu*))
        ((= contour 17)(box scale scale xcoord ycoord *yellow8-alu*))
        ((= contour 18)(box scale scale xcoord ycoord *yellow7-alu*))
        ((= contour 19)(box scale scale xcoord ycoord *yellow6-alu*))
        ((= contour 20)(box scale scale xcoord ycoord *yellow5-alu*))
        ((= contour 21)(box scale scale xcoord ycoord *orange9-alu*))))))
  (close input-stream))

(defun display-veg (mapsize classfile x-start y-start scale)
  (setq input-stream (open classfile :direction :input))
  (box (+ (* scale mapsize) 20) (+ (* scale mapsize) 20) x-start y-start *black-alu*)
  (cond ((and (= mapsize 80) (> scale 5))
         (display-scale-veg (+ (+ (* scale mapsize) 20) x-start) y-start scale
                  (+ 10 (* scale 18)) (+ (* scale mapsize) 20))))
  (do ((xcoord (+ x-start 10) (+ xcoord scale)))
     ((= xcoord (+ (* scale mapsize) (+ x-start 10))))
    (do ((ycoord (+ (* scale (1- mapsize)) (+ y-start 10)) (- ycoord scale)))
       ((= ycoord (+ y-start (- 10 scale))))
      (let ((veg (read input-stream)))
        (cond
          ((= veg 0)(box scale scale xcoord ycoord *tan-alu*))
          ((= veg 1)(box scale scale xcoord ycoord *green5-alu*))
          ((= veg 2)(box scale scale xcoord ycoord *green6-alu*))
          ((= veg 3)(box scale scale xcoord ycoord *green7-alu*))
          ((= veg 4)(box scale scale xcoord ycoord *green8-alu*))
          ((= veg 5)(box scale scale xcoord ycoord *green9-alu*))
          ((= veg 6)(box scale scale xcoord ycoord *orange-alu*))
          ((= veg 7)(box scale scale xcoord ycoord *red-alu*))))))
  (close input-stream))


(defun display-scale-class (x-start y-start scale b-width b-height)
  (box (+ b-width 10) b-height x-start y-start *black-alu*)
  (send *blue-window* :draw-string "   CLASS" (+ x-start 12) (+ (* scale 3) y-start))
  (box b-width (- (* scale 3) 1) x-start (+ (* scale 7) y-start) *green-alu*)
  (send *blue-window* :draw-string "Flat" (+ x-start 12) (+ (* scale 9) y-start)
              (+ x-start 12) (+ (* scale 9) y-start) nil nil *black-alu*)
  (box b-width (- (* scale 3) 1) x-start (+ (* scale 10) y-start) *black-alu*)
  (send *blue-window* :draw-string "Peak" (+ x-start 12) (+ (* scale 12) y-start))
  (box b-width (- (* scale 3) 1) x-start (+ (* scale 13) y-start) *black1-alu*)
  (send *blue-window* :draw-string "Ridge" (+ x-start 12) (+ (* scale 15) y-start))
```

```
(box b-width (- (* scale 3) 1) x-start (+ (* scale 16) y-start) *black2-alu*)
(send *blue-window* :draw-string "Saddle" (+ x-start 12) (+ (* scale 18) y-start)
        (+ x-start 12) (+ (* scale 18) y-start) nil nil *black-alu*)
(box b-width (- (* scale 3) 1) x-start (+ (* scale 19) y-start) *purple-alu*)
(send *blue-window* :draw-string "R-l Ridge" (+ x-start 12) (+ (* scale 21) y-start))
(box b-width (- (* scale 3) 1) x-start (+ (* scale 22) y-start) *blue-alu*)
(send *blue-window* :draw-string "Pit" (+ x-start 12) (+ (* scale 24) y-start))
(box b-width (- (* scale 3) 1) x-start (+ (* scale 25) y-start) *blue1-alu*)
(send *blue-window* :draw-string "Ravine" (+ x-start 12) (+ (* scale 27) y-start))
(box b-width (- (* scale 3) 1) x-start (+ (* scale 28) y-start) *blue2-alu*)
(send *blue-window* :draw-string "Pass" (+ x-start 12) (+ (* scale 30) y-start))
(box b-width (- (* scale 3) 1) x-start (+ (* scale 31) y-start) *blugre-alu*)
(send *blue-window* :draw-string "R-l Ravine" (+ x-start 12) (+ (* scale 33) y-start))
(box b-width (- (* scale 3) 1) x-start (+ (* scale 34) y-start) *yel1-alu*)
(send *blue-window* :draw-string "Planar  2-5" (+ x-start 12) (+ (* scale 36) y-start)
        (+ x-start 12) (+ (* scale 36) y-start) nil nil *black-alu*)
(box b-width (- (* scale 3) 1) x-start (+ (* scale 37) y-start) *yel2-alu*)
(send *blue-window* :draw-string "Planar  5-9" (+ x-start 12) (+ (* scale 39) y-start)
        (+ x-start 12) (+ (* scale 39) y-start) nil nil *black-alu*)
(box b-width (- (* scale 3) 1) x-start (+ (* scale 40) y-start) *yel3-alu*)
(send *blue-window* :draw-string "Planar  9-13" (+ x-start 12) (+ (* scale 42) y-start)
        (+ x-start 12) (+ (* scale 42) y-start) nil nil *black-alu*)
(box b-width (- (* scale 3) 1) x-start (+ (* scale 43) y-start) *yel4-alu*)
(send *blue-window* :draw-string "Planar 13-17" (+ x-start 12) (+ (* scale 45) y-start)
        (+ x-start 12) (+ (* scale 45) y-start) nil nil *black-alu*)
(box b-width (- (* scale 3) 1) x-start (+ (* scale 46) y-start) *yel5-alu*)
(send *blue-window* :draw-string "Planar 17-21" (+ x-start 12) (+ (* scale 48) y-start)
        (+ x-start 12) (+ (* scale 48) y-start) nil nil *black-alu*)
(box b-width (- (* scale 3) 1) x-start (+ (* scale 49) y-start) *yel6-alu*)
(send *blue-window* :draw-string "Planar 21-25" (+ x-start 12) (+ (* scale 51) y-start)
        (+ x-start 12) (+ (* scale 51) y-start) nil nil *black-alu*)
(box b-width (- (* scale 3) 1) x-start (+ (* scale 52) y-start) *yel7-alu*)
(send *blue-window* :draw-string "Planar 25-28" (+ x-start 12) (+ (* scale 54) y-start)
        (+ x-start 12) (+ (* scale 54) y-start) nil nil *black-alu*)
(box b-width (- (* scale 3) 1) x-start (+ (* scale 55) y-start) *red-alu*)
(send *blue-window* :draw-string "Unsafe" (+ x-start 12) (+ (* scale 57) y-start))
(box b-width (- (* scale 3) 1) x-start (+ (* scale 58) y-start) *red8-alu*)
(send *blue-window* :draw-string "Edge" (+ x-start 12) (+ (* scale 60) y-start)))


(defun display-scale-slope (x-start y-start scale b-width b-height)
  (box (+ b-width 10) b-height x-start y-start *black-alu*)
  (send *blue-window* :draw-string "  SLOPE" (+ x-start 12) (+ (* scale 3) y-start))
  (box b-width (- (* scale 3) 1) x-start (+ (* scale 7) y-start) *green-alu*)
  (send *blue-window* :draw-string "Flat" (+ x-start 12) (+ (* scale 9) y-start)
```

```
            (+ x-start 12) (+ (* scale 9) y-start) nil nil *black-alu*)
  (box b-width (- (* scale 3) 1) x-start (+ (* scale 10) y-start) *yel1-alu*)
  (send *blue-window* :draw-string "Safe  2-5" (+ x-start 12) (+ (* scale 12) y-start)
            (+ x-start 12) (+ (* scale 12) y-start) nil nil *black-alu*)
  (box b-width (- (* scale 3) 1) x-start (+ (* scale 13) y-start) *yel2-alu*)
  (send *blue-window* :draw-string "Safe  5-9" (+ x-start 12) (+ (* scale 15) y-start)
            (+ x-start 12) (+ (* scale 15) y-start) nil nil *black-alu*)
  (box b-width (- (* scale 3) 1) x-start (+ (* scale 16) y-start) *yel3-alu*)
  (send *blue-window* :draw-string "Safe  9-13" (+ x-start 12) (+ (* scale 18) y-start)
            (+ x-start 12) (+ (* scale 18) y-start) nil nil *black-alu*)
  (box b-width (- (* scale 3) 1) x-start (+ (* scale 19) y-start) *yel4-alu*)
  (send *blue-window* :draw-string "Safe 13-17" (+ x-start 12) (+ (* scale 21) y-start)
            (+ x-start 12) (+ (* scale 21) y-start) nil nil *black-alu*)
  (box b-width (- (* scale 3) 1) x-start (+ (* scale 22) y-start) *yel5-alu*)
  (send *blue-window* :draw-string "Safe 17-21" (+ x-start 12) (+ (* scale 24) y-start)
            (+ x-start 12) (+ (* scale 24) y-start) nil nil *black-alu*)
  (box b-width (- (* scale 3) 1) x-start (+ (* scale 25) y-start) *yel6-alu*)
  (send *blue-window* :draw-string "Safe 21-25" (+ x-start 12) (+ (* scale 27) y-start)
            (+ x-start 12) (+ (* scale 27) y-start) nil nil *black-alu*)
  (box b-width (- (* scale 3) 1) x-start (+ (* scale 28) y-start) *yel7-alu*)
  (send *blue-window* :draw-string "Safe 25-28" (+ x-start 12) (+ (* scale 30) y-start)
            (+ x-start 12) (+ (* scale 30) y-start) nil nil *black-alu*)
  (box b-width (- (* scale 3) 1) x-start (+ (* scale 31) y-start) *red-alu*)
  (send *blue-window* :draw-string "Unsafe" (+ x-start 12) (+ (* scale 33) y-start))
  (box b-width (- (* scale 3) 1) x-start (+ (* scale 34) y-start) *red8-alu*)
  (send *blue-window* :draw-string "Edge" (+ x-start 12) (+ (* scale 36) y-start)))


(defun display-scale-cont (x-start y-start scale b-width b-height)
  (box (+ b-width 10) b-height x-start y-start *black-alu*)
  (send *blue-window* :draw-string " ELEVATION" (+ x-start 12) (+ (* scale 3) y-start))
  (send *blue-window* :draw-string " SCALE(ft)" (+ x-start 12) (+ (* scale 6) y-start))
  (box b-width (- (* scale 3) 1) x-start (+ (* scale 7) y-start) *red-alu*)
  (send *blue-window* :draw-string "   <1000" (+ x-start 12) (+ (* scale 9) y-start))
  (box b-width (- (* scale 3) 1) x-start (+ (* scale 10) y-start) *blue9-alu*)
  (send *blue-window* :draw-string "1000-1040" (+ x-start 12) (+ (* scale 12) y-start))
  (box b-width (- (* scale 3) 1) x-start (+ (* scale 13) y-start) *blue8-alu*)
  (send *blue-window* :draw-string "1040-1080" (+ x-start 12) (+ (* scale 15) y-start))
  (box b-width (- (* scale 3) 1) x-start (+ (* scale 16) y-start) *blue7-alu*)
  (send *blue-window* :draw-string "1080-1120" (+ x-start 12) (+ (* scale 18) y-start))
  (box b-width (- (* scale 3) 1) x-start (+ (* scale 19) y-start) *blue6-alu*)
  (send *blue-window* :draw-string "1120-1160" (+ x-start 12) (+ (* scale 21) y-start))
  (box b-width (- (* scale 3) 1) x-start (+ (* scale 22) y-start) *blue5-alu*)
  (send *blue-window* :draw-string "1160-1200" (+ x-start 12) (+ (* scale 24) y-start)
            (+ x-start 12) (+ (* scale 24) y-start) nil nil *black-alu*)
```

```
(box b-width (- (* scale 3) 1) x-start (+ (* scale 25) y-start) *green9-alu*)
(send *blue-window* :draw-string "1200-1240" (+ x-start 12) (+ (* scale 27) y-start))
(box b-width (- (* scale 3) 1) x-start (+ (* scale 28) y-start) *green8-alu*)
(send *blue-window* :draw-string "1240-1280" (+ x-start 12) (+ (* scale 30) y-start))
(box b-width (- (* scale 3) 1) x-start (+ (* scale 31) y-start) *green7-alu*)
(send *blue-window* :draw-string "1280-1320" (+ x-start 12) (+ (* scale 33) y-start))
(box b-width (- (* scale 3) 1) x-start (+ (* scale 34) y-start) *green6-alu*)
(send *blue-window* :draw-string "1320-1360" (+ x-start 12) (+ (* scale 36) y-start))
(box b-width (- (* scale 3) 1) x-start (+ (* scale 37) y-start) *green5-alu*)
(send *blue-window* :draw-string "1360-1400" (+ x-start 12) (+ (* scale 39) y-start)
        (+ x-start 12) (+ (* scale 39) y-start) nil nil *black-alu*)
(box b-width (- (* scale 3) 1) x-start (+ (* scale 40) y-start) *red9-alu*)
(send *blue-window* :draw-string "1400-1440" (+ x-start 12) (+ (* scale 42) y-start))
(box b-width (- (* scale 3) 1) x-start (+ (* scale 43) y-start) *red8-alu*)
(send *blue-window* :draw-string "1440 1480" (+ x-start 12) (+ (* scale 45) y-start))
(box b-width (- (* scale 3) 1) x-start (+ (* scale 46) y-start) *red7-alu*)
(send *blue-window* :draw-string "1480-1520" (+ x-start 12) (+ (* scale 48) y-start))
(box b-width (- (* scale 3) 1) x-start (+ (* scale 49) y-start) *red6-alu*)
(send *blue-window* :draw-string "1520-1560" (+ x-start 12) (+ (* scale 51) y-start))
(box b-width (- (* scale 3) 1) x-start (+ (* scale 52) y-start) *red5-alu*)
(send *blue-window* :draw-string "1560-1600" (+ x-start 12) (+ (* scale 54) y-start)
        (+ x-start 12) (+ (* scale 54) y-start) nil nil *black-alu*)
(box b-width (- (* scale 3) 1) x-start (+ (* scale 55) y-start) *yellow9-alu*)
(send *blue-window* :draw-string "1600-1640" (+ x-start 12) (+ (* scale 57) y-start))
(box b-width (- (* scale 3) 1) x-start (+ (* scale 58) y-start) *yellow8-alu*)
(send *blue-window* :draw-string "1640-1680" (+ x-start 12) (+ (* scale 60) y-start))
(box b-width (- (* scale 3) 1) x-start (+ (* scale 61) y-start) *yellow7-alu*)
(send *blue-window* :draw-string "1680-1720" (+ x-start 12) (+ (* scale 63) y-start))
(box b-width (- (* scale 3) 1) x-start (+ (* scale 64) y-start) *yellow6-alu*)
(send *blue-window* :draw-string "1720-1760" (+ x-start 12) (+ (* scale 66) y-start))
(box b-width (- (* scale 3) 1) x-start (+ (* scale 67) y-start) *yellow5-alu*)
(send *blue-window* :draw-string "1760-1800" (+ x-start 12) (+ (* scale 69) y-start)
        (+ x-start 12) (+ (* scale 69) y-start) nil nil *black-alu*)
(box b-width (- (* scale 3) 1) x-start (+ (* scale 70) y-start) *orange9-alu*)
(send *blue-window* :draw-string "    >1800" (+ x-start 12) (+ (* scale 72) y-start)))


(defun display-scale-veg (x-start y-start scale b-width b-height)
 (box (+ b-width 10) b-height x-start y-start *black-alu*)
 (send *blue-window* :draw-string "VEGETATION" (+ x-start 12) (+ (* scale 3) y-start))
 (send *blue-window* :draw-string "HEIGHTS(m)" (+ x-start 12) (+ (* scale 6) y-start))
 (box b-width (- (* scale 3) 1) x-start (+ (* scale 7) y-start) *tan-alu*)
 (send *blue-window* :draw-string "   0-1" (+ x-start 12) (+ (* scale 9) y-start)
        (+ x-start 12) (+ (* scale 9) y-start) nil nil *black-alu*)
 (box b-width (- (* scale 3) 1) x-start (+ (* scale 10) y-start) *green5-alu*)
```

```
  (send *blue-window* :draw-string "   1-4" (+ x-start 12) (+ (* scale 12) y-start)
                (+ x-start 12) (+ (* scale 12) y-start) nil nil *black-alu*)
  (box b-width (- (* scale 3) 1) x-start (+ (* scale 13) y-start) *green6-alu*)
  (send *blue-window* :draw-string "   4-8" (+ x-start 12) (+ (* scale 15) y-start))
  (box b-width (- (* scale 3) 1) x-start (+ (* scale 16) y-start) *green7-alu*)
  (send *blue-window* :draw-string "   8-12" (+ x-start 12) (+ (* scale 18) y-start))
  (box b-width (- (* scale 3) 1) x-start (+ (* scale 19) y-start) *green8-alu*)
  (send *blue-window* :draw-string "  12-20" (+ x-start 12) (+ (* scale 21) y-start))
  (box b-width (- (* scale 3) 1) x-start (+ (* scale 22) y-start) *green9-alu*)
  (send *blue-window* :draw-string "    >20" (+ x-start 12) (+ (* scale 24) y-start))
  (box b-width (- (* scale 3) 1) x-start (+ (* scale 25) y-start) *orange-alu*)
  (send *blue-window* :draw-string " No Data" (+ x-start 12) (+ (* scale 27) y-start))
  (box b-width (- (* scale 3) 1) x-start (+ (* scale 28) y-start) *red-alu*)
  (send *blue-window* :draw-string " Not Used" (+ x-start 12) (+ (* scale 30) y-start)))


(defun demo-fq5580 ()
  (send *blue-window* :refresh)
  (display-class 80 "fq5580.class" 5 0 6)
  (display-slope 80 "fq5580.slope" 5 505 6)
  (display-cont 80 "fq5580.cont" 640 0 6)
  (display-veg 80 "fq5580.1-hr-v" 640 505 6))


(defun demo-fq5886 ()
  (send *blue-window* :refresh)
  (display-class 80 "fq5886.class" 5 0 6)
  (display-slope 80 "fq5886.slope" 5 505 6)
  (display-cont 80 "fq5886.cont" 640 0 6)
  (display-veg 80 "fq5886.1-hr-v" 640 505 6))


(defun demo-fq5889 ()
  (send *blue-window* :refresh)
  (display-class 80 "fq5889.class" 5 0 6)
  (display-slope 80 "fq5889.slope" 5 505 6)
  (display-cont 80 "fq5889.cont" 640 0 6)
  (display-veg 80 "fq5889.1-hr-v" 640 505 6))



(defun final-demo-fq5580 ()
  (send *blue-window* :refresh)
  (display-class 80 "fq5580.class" 5 0 6)
  (display-class 80 "fq5580.class-sm" 5 505 6)
```

```
  (display-cont 80 "fq5580.cont" 640 0 6)
  (display-cont 80 "fq5580.cont-sm" 640 505 6))


(defun final-demo-fq5886 ()
  (send *blue-window* :refresh)
  (display-class 80 "fq5886.class" 5 0 6)
  (display-class 80 "fq5886.class-sm" 5 505 6)
  (display-cont 80 "fq5886.cont" 640 0 6)
  (display-cont 80 "fq5886.cont-sm" 640 505 6))


(defun final-demo-fq5889 ()
  (send *blue-window* :refresh)
  (display-class 80 "fq5889.class" 5 0 6)
  (display-class 80 "fq5889.class-sm" 5 505 6)
  (display-cont 80 "fq5889.cont" 640 0 6)
  (display-cont 80 "fq5889.cont-sm" 640 505 6))

(defun compare-fq5580 ()
  (send *blue-window* :refresh)
  (display-class 80 "fq5580.class" 5 0 6)
  (display-class 80 "fq5580.class-sm" 5 505 6)
  (display-class 80 "fq5580.class-ch" 640 0 6)
  (display-cont 80 "fq5580.cont" 640 505 6))


(defun compare-fq5886 ()
  (send *blue-window* :refresh)
  (display-class 80 "fq5886.class" 5 0 6)
  (display-class 80 "fq5886.class-sm" 5 505 6)
  (display-class 80 "fq5886.class-ch" 640 0 6)
  (display-cont 80 "fq5886.cont" 640 505 6))


(defun compare-fq5889 ()
  (send *blue-window* :refresh)
  (display-class 80 "fq5889.class" 5 0 6)
  (display-class 80 "fq5889.class-sm" 5 505 6)
  (display-class 80 "fq5889.class-ch" 640 0 6)
  (display-cont 80 "fq5889.cont" 640 505 6))
```

```
(defun demo-classes ()
  (send *blue-window* :refresh)
  (box 160 1000 1090 10 *black-alu*)
  (display-scale-class 1100 10 15 150 1000)
  (display-class 7 "test-level.class" 100 100 20)
  (display-class 7 "test-planar-10.class" 350 100 20)
  (display-class 7 "test-planar-20.class" 600 100 20)
  (display-class 7 "test-planar-30.class" 850 100 20)
  (display-class 7 "test-peak.class" 100 400 20)
  (display-class 7 "test-ridge.class" 350 400 20)
  (display-class 7 "test-ridge-diag.class" 600 400 20)
  (display-class 7 "test-pit.class" 100 700 20)
  (display-class 7 "test-ravine.class" 350 700 20)
  (display-class 7 "test-ravine-diag.class" 600 700 20))


(defun demo-class ()
  (send *blue-window* :refresh)
  (display-class 80 "fq5580.class" 5 0 6)
  (display-class 80 "fq5886.class" 5 505 6)
  (display-class 80 "fq5889.class" 640 0 6))


(defun demo-slope ()
  (send *blue-window* :refresh)
  (display-slope 80 "fq5580.slope" 5 0 6)
  (display-slope 80 "fq5886.slope" 5 505 6)
  (display-slope 80 "fq5889.slope" 640 0 6))


(defun demo-cont ()
  (send *blue-window* :refresh)
  (display-cont 80 "fq5580.cont" 5 0 6)
  (display-cont 80 "fq5886.cont" 5 505 6)
  (display-cont 80 "fq5889.cont" 640 0 6))


(defun demo-veg ()
  (send *blue-window* :refresh)
  (display-veg 80 "fq5580.1-hr-v" 5 0 6)
  (display-veg 80 "fq5886.1-hr-v" 5 505 6)
  (display-veg 80 "fq5889.1-hr-v" 640 0 6))
```

```
(defun show-class-fq5886 ()
  (send *blue-window* :refresh)
  (display-class 80 "fq5886.class" 0 0 5)
  (send *blue-window* :draw-string "Unsmoothed  Curve .01  Dot .01" 50 450)
  (display-class 80 "fq5886.class-sm" 0 500 5)
  (send *blue-window* :draw-string "Smoothed  Curve .01  Dot .01" 50 950)
  (display-class 80 "fq5886.class-tst1" 425 0 5)
  (send *blue-window* :draw-string "Smoothed  Curve .006  Dot .001" 500 450)
  (display-class 80 "fq5886.class-ch" 850 0 5)
  (send *blue-window* :draw-string "Smoothed  Curve .006  Dot .01" 900 450)
  (display-class 80 "fq5886.class-tst2" 425 500 5)
  (send *blue-window* :draw-string "Smoothed  Curve .004  Dot .001" 500 950)
  (display-class 80 "fq5886.class-tst3" 850 500 5)
  (send *blue-window* :draw-string "Smoothed  Curve .004  Dot .01" 900 950))
```

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943-5002	2
3.	Chief of Naval Operations Director, Information Systems (OP-945) Navy Department Washington, D.C. 20350-2000	1
4.	US Army Combat Developments Experimentation Center (USACDEC) Attention: W. D. West Fort Ord, California 93941	1
5.	Department Chairman, Code 52 Department of Computer Science Naval Postgraduate School Monterey, California 93943-5000	2
6.	Curriculum Officer, Code 37 Computer Technology Naval Postgraduate School Monterey, California 93943-5000	1
7.	Professor Robert B. McGhee, Code 52MZ Computer Science Department Naval Postgraduate School Monterey, California 93943-5000	11
8.	Professor Michael J. Zyda, Code 52ZK Computer Science Department Naval Postgraduate School Monterey, California 93943-5000	1

9.	Professor Neil C. Rowe, Code 52RP Computer Science Department Naval Postgraduate School Monterey, California 93943-5000	1
10.	Commandant of the Marine Corps Code TE 06 Headquarters, U.S. Marine Corps Washington, D.C. 20360-0001	1
11.	Naval Military Personnel Center.922C ATTN: LT Goodpasture 1300 Wilson Blvd Rosslyn, Virginia 22209	1
12.	United States Military Academy Department of Geography & Computer Science ATTN: MAJ R. F. Richbourg West Point, New York 10996-1695	1
13.	Artificial Intelligence Center HQDA OCSA ATTN: DACS-DMA The Pentagon RM 1D659 Washington, D.C. 20310-0200	1
14.	Director of Research Administration, Code 012 Naval Postgraduate School Monterey, California 93943	1
15.	Mr & Mrs Henry Felhoelter 200 Redding Road Louisville, Kentucky 40218	2
16.	Mr & Mrs Bernard Mehringer 3110 Leslie Drive Jasper, Indiana 47546	1